UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Departamento de Engenharia Elétrica

Pós-Graduação em Engenharia Elétrica

Sistemas em Tempo Real

DMA e Interrupções

Alunos:
Adriano de Oliveira Andrade
Anderson Luiz Aguiar Vilaça
Luciane Dias Silva

Data: 04/11/98

ÍNDICE

1 - INTRODUÇÃO

Este trabalho tem por objetivo apresentar o hardware desenvolvido para transferências de dados através dos canais de DMA e de interrupções.

Para uma melhor compreensão do trabalho desenvolvido, este relatório foi divido em 4 partes a saber:

1 – Apresentação teórica do funcionamento do canais de DMA;
2 – Apresentação teórica do funcionamento das Interrupções;
3 – Descrição do hardware desenvolvido para o trabalho;
4 – Apresentação e descrição do software desenvolvido para manipular as transferências através do DMA e de Interrupções.

2 - DMA

Em muitos casos, é necessário transferir dados a uma velocidade mais alta que a oferecida pelo método de I/O (com instruções IN e OUT). Um bom exemplo disso é a transferência de dados feita pelos discos flexíveis. A velocidade da transferência realizada pelo adaptador de disco é alta o suficiente para tornar difícil para a CPU atender a essa transferência. Para resolver estes problemas de transferências de dados a alta velocidade, existe no PC uma função especial, chamada de acesso direto à memória (DMA – *Direct Memory Access*). Através do controlador de DMA, uma interface pode ler ou escrever na memória, sem a intervenção da CPU. No projeto do PC XT, foi usado o controlador de DMA 8237–5. A partir do PC AT, usam-se dois 8237 em cascata, sendo que o segundo realiza transferência de dados em dezesseis bits.

Durante a execução dos programas a CPU gera o endereço e os sinais de controle para enviar ou receber um dado da memória. Quando uma interface necessita de realizar uma transferência usando DMA, ela envia um pedido ao controlador de DMA. O controlador, de acordo com as prioridades, envia um pedido de HOLD para a CPU. Ao final do ciclo de barramento que estava em andamento, a CPU libera o barramento e envia um reconhecimento de HOLD (HLDA) para o controlador de DMA, indicando que o barramento está livre. O controlador, então, toma posse do barramento e envia o endereço da memória e os sinais de controle de forma a realizar a transferência do dado entre a memória e a interface. A interface é alertada de que o pedido de DMA vai ser realizado, através da linha de reconhecimento denominada DACK (gerada pelo controlador de DMA).

O controlador de DMA pode ser encarado como um terceiro elemento que, quando solicitado, assume o controle do barramento e realiza transferência de dados entre a interface e a memória. Durante as operações de DMA, o dado é transferido diretamente entre a interface e a memória, como ilustrado na figura 1.

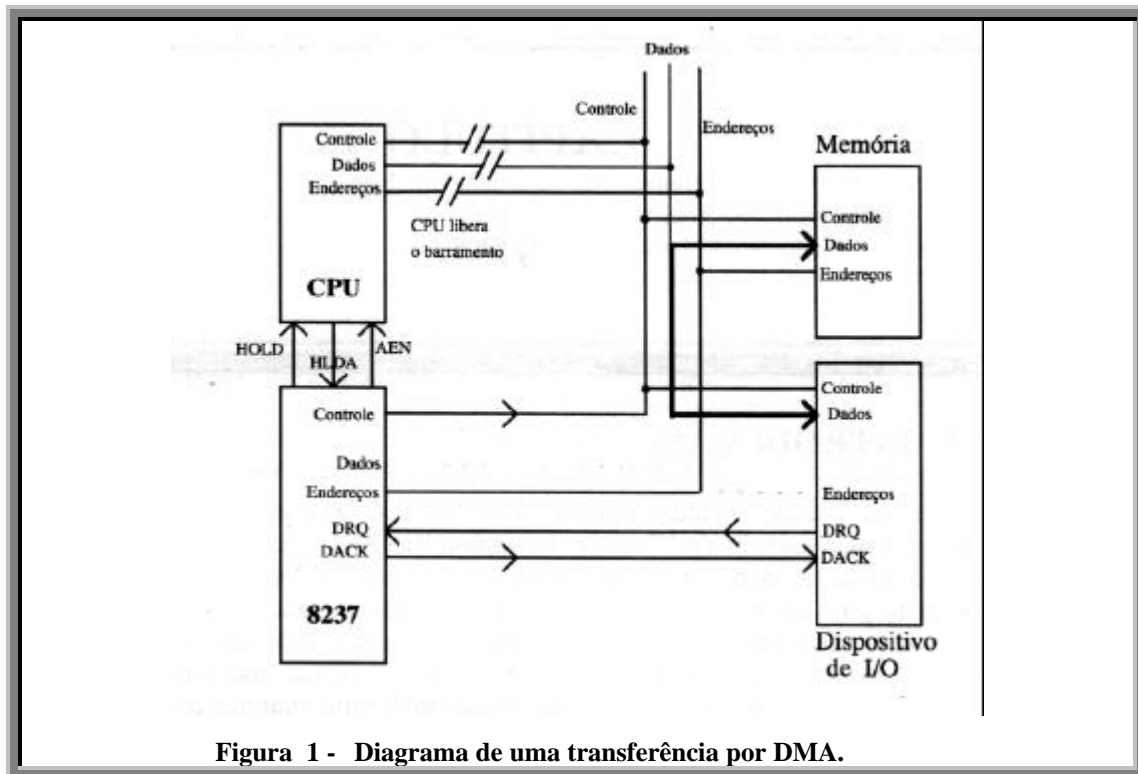



Figura 1 - Diagrama de uma transferência por DMA.

2.1 – Operação do DMA

A seguir, é feita uma descrição, passo a passo, do que ocorre durante um ciclo de DMA, que deve ser vista junto com a ilustração da figura 2.

1. Antes que uma operação de DMA possa ocorrer, o controlador 8237 deve ser inicializado (através de palavras de controle), de forma a realizar os ciclos apropriados. Os seguintes itens são necessários para a inicialização:
    - selecionar o tipo de função: leitura ou escrita na memória.
    - selecionar o tipo de transferência: rajada ou um único byte (ou word – 16 bits).
    - especificar a quantidade de bytes (ou words) a serem transferidos.
    - indicar a prioridade dos canais.
    - especificar o endereço inicial da memória.
    - habilitar os canais a serem usados.

2. A interface, que precisa realizar um ciclo de DMA, envia um sinal DRQ (*DMA request*) para o controlador 8237, indicando o pedido em um determinado canal.
3. O 8237 resolve a prioridade deste pedido, junto com as dos outros canais, e envia um HRQ (*hold request*) para o circuito gerador de estados de espera do PC.
4. O circuito gerador de estados de espera monitora as linhas de estado da CPU, procurando um estado passivo, ou seja, procura um instante em que o barramento esteja inativo ou próximo do final de um ciclo.
5. Quando um estado passivo é detectado, é enviado um sinal *not ready* para a CPU, fazendo com que a mesma entre em estado de espera. Além disso, um sinal de HLDA (*hold acknowledge*) é enviado para o 8237, indicando que, no próximo pulso de *clock*, o barramento estará livre e que o ciclo de DMA poderá ocorrer. São enviados sinais para colocar em alta impedância os *buffers* que conectam a CPU ao barramento (isso desconecta a CPU do barramento).
6. O controlador de DMA, ao receber o HLDA, envia um sinal DACK para a interface que solicitou o DMA. Esse sinal atua como o selecionador do chip (*chip select*), conectando a interface ao barramento de dados do PC.
7. Agora, o 8237 coloca no barramento o endereço da memória onde acontecerá a operação de DMA. Em seguida, envia os sinais de controle (MEMR e IOW) ou (MEMW e IOR), de forma a realizar o ciclo programado.
8. Após receber o DACK, a interface retira o pedido de DMA que havia feito através da linha DRQ. Quando o controlador termina o ciclo, ele retira o pedido HRQ que enviou para o circuito gerador de estados de espera, que, por sua vez, remove o sinal HLDA, indicando que a CPU irá assumir o controle do barramento. Finalmente, o circuito gerador de estado de espera retira o sinal *not ready* e habilita os *buffers* da CPU. Assim, o ciclo de barramento pode ser terminado.

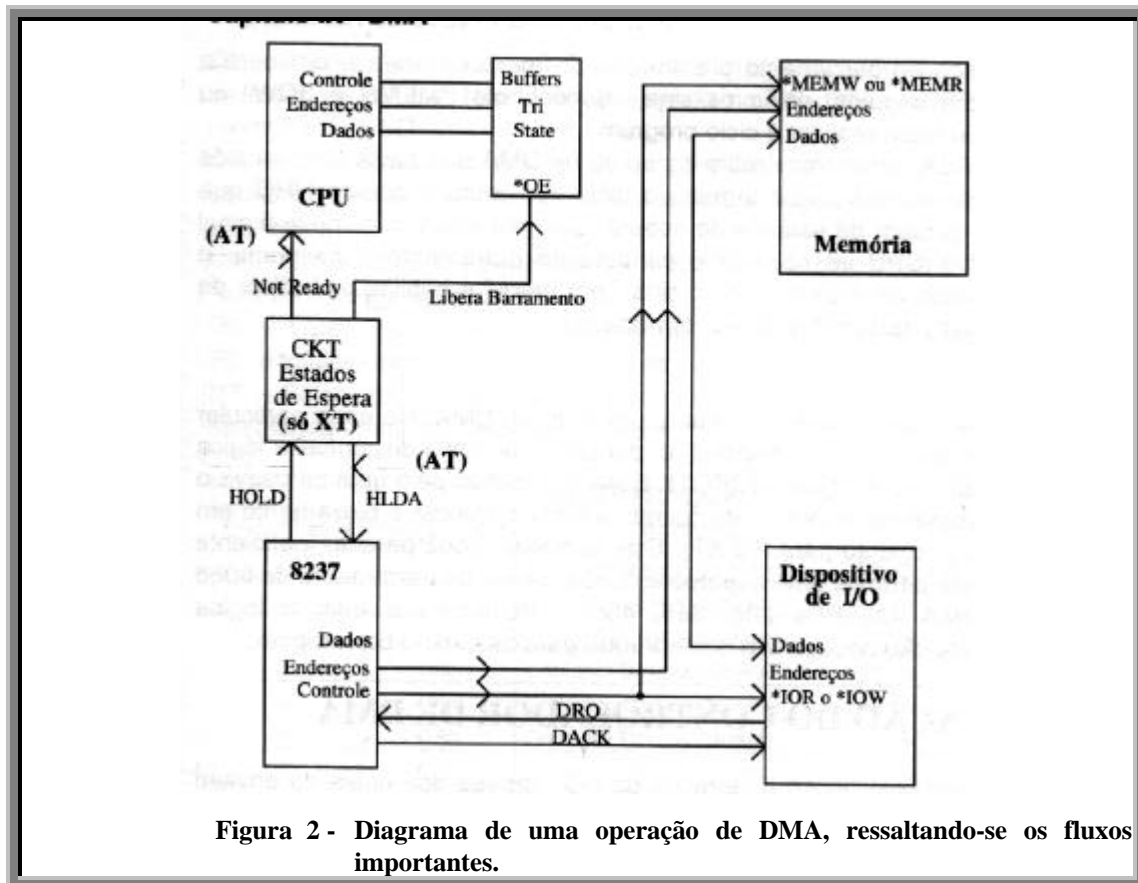



Figura 2 - Diagrama de uma operação de DMA, ressaltando-se os fluxos importantes.

OBS: Todas essas operações são realizadas durante um ciclo de DMA. No caso particular do PC XT, a CPU operava em modo máximo, e, por isso, não tinha disponível a lógica HOLD/HLDA (existe apenas a lógica RQ/GT). Esse é o motivo pelo qual se usava o circuito gerador de estados de espera para “parar” a CPU e colocar o barramento em alta impedância (itens 3,4,5 somente para PC XT). Os PCs 286, 386, 486 e Pentium possuem a lógica HOLD/HLDA e, por isso, não necessitam deste artifício para assumir o barramento.

2.2 – Inicialização do Controlador de DMA

O controlador 8237 necessita de 16 endereços de I/O, através dos quais se enviam dados de inicialização e lê-se o estado do controlador. Escreve-se nos dezesseis endereços reservados para ele, porém, nem todos os endereços podem ser lidos. O projeto do PC mapeia o primeiro controlador 8237 nos endereços de I/O que vão de 0 a Fh. O segundo 8237, que só existe nos 286, 386, 486 e Pentium, ocupa os endereços

de C0h a DFh. A tabela 1 descreve os endereços de I/O relacionados às operações de DMA de um PC/XT (que possui apenas um 8237). Os bits de endereços A16-A19 são relacionados pelos registradores de paginação, externos ao 8237.

Tabela 1 - Mapa de endereços ocupados pelo 8237 num PC/XT.

Endereço	Operação	Acesso
00h	escrita	registradores de endereço base e corrente do canal 0
00h	leitura	registrador de endereço corrente do canal 0
01h	escrita	registradores de palavra corrente e contador de palavra base do canal 0
01h	leitura	registrador de palavra corrente do canal 0
02h	escrita	registradores de endereço base e corrente do canal 1
02h	leitura	registrador de endereço corrente do canal 1
03h	escrita	registradores de palavra corrente e contador de palavra base do canal 1
03h	leitura	registrador de palavra corrente do canal 1
04h	escrita	registradores de endereço base e corrente do canal 2
04h	leitura	registrador de endereço corrente do canal 2
05h	escrita	registradores de palavra corrente e contador de palavra base do canal 2
05h	leitura	registrador de palavra corrente do canal 2
06h	escrita	registradores de endereço base e corrente do canal 3
06h	leitura	registrador de endereço corrente do canal 3
07h	escrita	registradores de palavra corrente e contador de palavra base do canal 3
07h	leitura	registrador de palavra corrente do canal 3
08h	leitura	registrador de status
08h	escrita	registrador de comando
09h	escrita	registrador de aquisição
0Ah	escrita	escreve um bit do registrador de mascaramento
0Bh	escrita	registrador de modo
0Ch	escrita	resseta flip-flop interno
0Dh	leitura	registrador temporário
0Dh	escrita	*master reset* – reinicializa 8237
0Eh	escrita	resseta registrador de mascaramento
0Fh	escrita	registrador de mascaramento
83h	leitura/escrita	registrador de paginação do canal 1
81h	leitura/escrita	registrador de paginação do canal 2
82h	leitura/escrita	registrador de paginação do canal 3

Nos controladores sucessores ao 8237, embora a idéia base do 8237 tenha sido mantida, as necessidades de ampliação do número de canais e das capacidades de transferência e endereçamento fez com que se aperfeiçoasse o projeto inicial do primeiro controlador de DMA.

Surgiram outros compatíveis que, além de ter o controlador de DMA integrado juntamente com outros circuitos (ex: controlador de *refresh*), incorporaram uma arquitetura que consiste basicamente de dois 8237 melhorados operando em cascata. O controlador de DMA melhorado possui como principais características:

- 8 canais de DMA;
- Endereçamento a 24 bits (capacidade de 16 MB);
- 4 canais com transferências de dados de 16 bits;
- Registradores de paginação internos de 8 bits, contra 4 bits no PC/XT;
- Compatibilidade com programas escritos para o 8237 (PC/XT).

A tabela 2 (a) e (b) complementa a tabela 1, estendendo os endereços de I/O necessários às operações de DMA de um PC/AT e citando o mapeamento padrão nos canais de DMA.

Uma importante mudança na decodificação dos endereços de I/O do segundo 8237 foi introduzida: cada registrador é acessado em dois endereços consecutivos, devido à não existência da linha A0. A linha A16 da CPU corresponderá a linha A15 do segundo 8237 e assim por diante.

Tabela 2 - Mapa de I/O para operações de DMA (a) e canais de DMA do PC (b).

Endereços	**Descrição**
00h a 0Fh	primeiro 8237, canais 0 a 3
C0h a CFh	segundo 8237, canais 4 a 7 (só no AT)
C0h ou C1h	registradores de endereço base e corrente do canal 4
C2h ou C3h	registradores de palavra corrente e contador de palavra base do canal 4
C4h ou C5h	registradores de endereço base e corrente do canal 5
C6h ou C7h	registradores de palavra corrente e contador de palavra base do canal 5
C8h ou C9h	registradores de endereço base e corrente do canal 6
CAh ou CBh	registradores de palavra corrente e contador de palavra base do canal 6
CCh ou CDh	registradores de endereço base e corrente do canal 7
CEh ou CFh	registradores de palavra corrente e contador de palavra base do canal 7
D0h ou D1h	registrador de status e registrador de comando do segundo 8237
D2h ou D3h	registrador de requisição do segundo 8237
D4h ou D5h	escrita de um bit do registrador de mascaramento do segundo 8237
D6h ou D7h	registrador de modo do segundo 8237
D8h ou D9h	resseta flip-flop interno do segundo 8237
DAh ou DBh	registrador temporário e *master reset* do segundo 8237
DCh ou DDh	resseta registrador de mascaramento do segundo 8237
DEh ou DFh	registrador de mascaramento do segundo 8237
87h	registrador de paginação do canal 0 (só no AT)
83h	registrador de paginação do canal 1
81h	registrador de paginação do canal 2
82h	registrador de paginação do canal 3
8Bh	registrador de paginação do canal 5 (só no AT)
89h	registrador de paginação do canal 6 (só no AT)
8Ah	registrador de paginação do canal 7 (só no AT)
8Fh	registrador de paginação para o *refresh*

(a)

Canal	**XT/AT**	**bits p/ canal**	**Usuário**
0	XT e AT	8	*refresh* da DRAM (XT)
1	XT e AT	8	SDLC
2	XT e AT	8	disco flexível
3	XT e AT	8	disponível
4	AT	16	cascata
5	AT	16	disponível
6	AT	16	disponível
7	AT	16	disponível

(b)

A seguir, é apresentada uma descrição de alguns dos registradores das tabelas 1 e 2:

Registrador de comando
(*Command Register)*
ENDEREÇO: (8h/D0h)

Esse registrador é acessado como uma escrita de I/O no endereço 8h (D0h). A descrição dos seus bits é feita na figura 3.

bit 0 = acesso memória-memória - 0: desabilitado ou 1: habilitado
bit 1 = retenção do endereço do canal 0 - 0: desabilitado ou 1: habilitado
bit 2 = controlador - 0: desabilitado ou 1: habilitado
bit 3 = temporização - 0: normal ou 1: comprimida
bit 4 = prioridade - 0: fixa ou 1: rotativa
bit 5 = deve ser igual a zero
bit 6 = nível de ativação para o sinal DREQ - 0: alto ou 1: baixo
bit 7 = nível de ativação para o sinal DACK - 0: alto ou 1: baixo
obs: - os bits1 e 3não afetam a operação se o bit 0 for 0.
- o bit 5 não afeta a operação se o bit 3 for

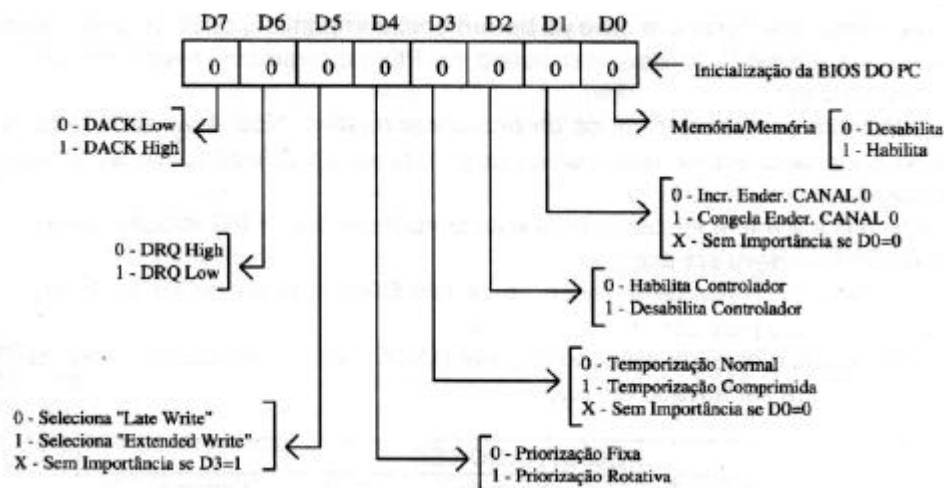



Figura 3 – Registrador de comando

Registrador de pedido (*Write Request Register)* ENDEREÇO: (9h/D2h)	Uma escrita no endereço de I/O 9h (D2h) faz com que se acesse o Registrador de Pedido. Com esse registrador, pode-se gerar pedidos de DMA por software. A figura 4 apresenta o significado dos bits desse registrador.

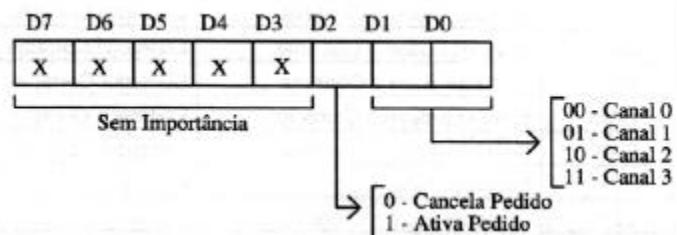



Figura 4 – Registrador de pedido

Registrador de Modo
(*Mode Register)*
ENDEREÇO: (Bh/D6h)

Esse registrador é acessado através do endereço de I/O Bh (D6h), e define os modos de operação para cada um dos quatro canais de DMA. A definição dos bits desse registrador é apresentada na figura 5.

bits 1 e 0 = seleção do canal - 00: canal 0, 01: canal 1, 10: canal 2 e 11:canal 3

bits 3 e 2 = 00: operação de verificação (lê e escreve dado - *refresh*), 01: lê dado da interface e escreve na memória , 10: lê dado da memória e escreve na interface

bit 4 = autoinicialização - 0: desativada ou 1: ativada

bit 5 = 0: incremento automático de endereços ou 1: decremento automático de endereços

bits 7 e 6 = 00: modo de transferência por demanda, 01: modo de transferência única, 10: modo de transferência por bloco, 11: modo cascata

Modo de Transferência Única - Neste modo, o dispositivo é programado para realizar a transferência de apenas um byte. O sinal DREQ deve ser mantido ativado (pelo canal solicitante) até que o sinal DACK (*DMA ACKnowledge*) seja ativado pelo 8237, reconhecendo o pedido. O sinal HRQ será desativado para que o 8237 comande os barramentos de dados e endereços do PC. A CPU reconhece a operação de DMA através do sinal HLDA (*Hold ACKnowledge*).

Modo de Transferência por Bloco - Neste modo, o dispositivo é programado para realizar transferências, continuamente, até que seja setado o sinal interno TC (*Terminal Count*) ou o sinal externo EOP (*End Of Process*). O sinal DREQ só precisa ser mantido ativado até que seja reconhecido o pedido pelo 8237 (DACK ativado).

Modo de Transferência por Demanda - Este modo é semelhante ao anterior. Faz transferências, continuamente, até ser interrompido pelos citados sinais TC ou EOP ou pela desativação de DREQ.

Modo Cascata - Este modo é utilizado no cascateamento de mais de um chip 8237. Um 8237 escravo (externo ao PC) interliga seus sinais HRQ e HLDA, respectivamente, aos sinais DREQ e DACK do 8237 (mestre), diretamente ligado à CPU, aumentando a capacidade de canais de DMA.

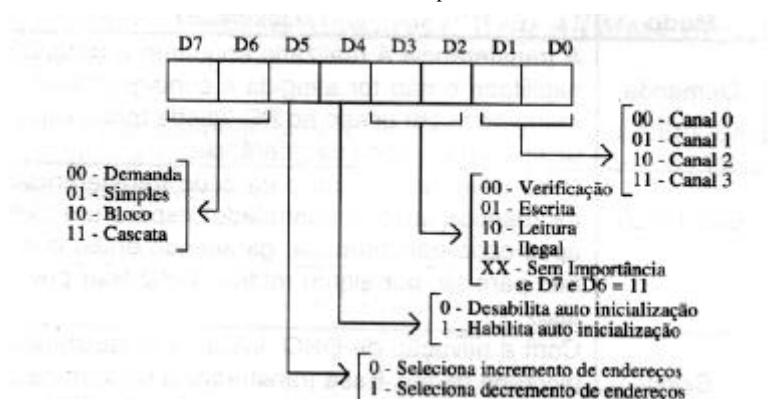



Figura 5 – Registrador de Modo

Registrador de Máscara Simples
(*Single Mask Bit)*
ENDEREÇO: (Ah/D4h)

Esse registrador é acessado através do endereço de I/O Ah (D4h). Ele permite ativar ou desativar, individualmente, a máscara de cada canal. Quando a máscara está ativada, a operação de DMA fica desabilitada e pode ser acessado de três formas. A figura 6 apresenta o significado dos bits desse registrador.

1. Pelo endereço 0Ah - seta ou resseta bit de máscara individualmente
   bits 1 e 0 = seleção do canal- 00: canal 0, 01: canal 1, 10: canal 2 e 11:canal 3
   bit 2 = 0: resseta bit de máscara, desabilitando o canal ou 1: seta bit de máscara, habilitando o canal
2. Pelo endereço 0Fh - escreve toda a máscara
   bit 0 = máscara do canal 0, bit 1 = máscara do canal 1
   bit 2 = máscara do canal 2, bit 3 = máscara do canal 3
3. É ressetado através de uma escrita no endereço 0Eh.

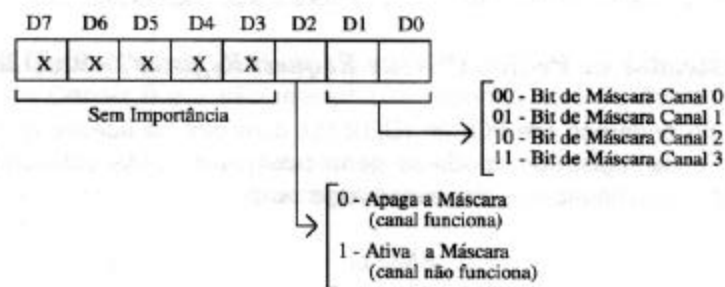



Figura 6 – Registrador de Máscara Simples

Registrador Apaga Máscara
(*Clear Mask Register)*
ENDEREÇO: (Eh/DCh)

Essa função é realizada quando se escreve no endereço de I/O Eh (DCh). Ela apaga todas as máscaras, habilitando então os quatro canais de DMA.

Registrador Escrita de Máscaras (*Write All Mask-Register Bits)* ENDEREÇO: (Fh/DEh)	Esse registrador é acessado através do endereço de I/O Fh (DEh). Ele é usado para, de forma individual e simultânea, controlar os bits de máscara dos canais de DMA. A figura 7 apresenta seus bits.

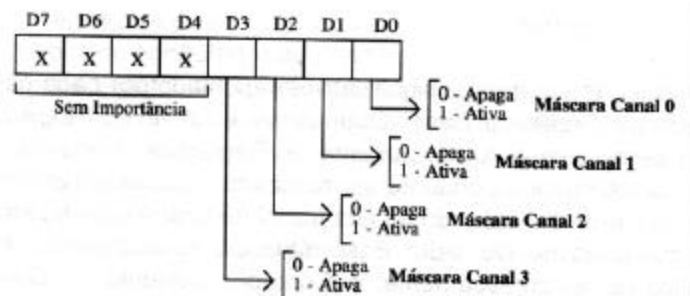



Figura 7 – Registrador de Escrita de Máscaras

Apaga Flip-Flop Apontador de Byte (*Clear Byte Pointer Flip-Flop)* ENDEREÇO: (Ch/D8h)	Uma escrita nesse endereço apenas apaga um flip-flop interno que é usado para selecionar o byte alto ou o byte baixo das palavras de 16 bits, isto nas escritas e leituras feitas nos endereços de 0h até 7h (C0h a CEh). Quando o flip-flop está apagado, a próxima operação de leitura ou escrita transfere o byte menos significativo da palavra de 16 bits. Esta operação de leitura ou escrita inverte o estado do flip-flop, que passa a indicar que se vai operar com o byte mais significativo (após essa operação ele será novamente invertido). Assim, com um barramento de oito bits, pode-se enviar ou receber valores de dezesseis bits. Ele é usado por ocasião da escrita nos Registradores de Endereço Base e Corrente, nos Contadores Base e Corrente, e também quando se lêem os Registradores de Endereço Corrente e Contador Corrente.

Zeragem Mestre (*Master Clear)* ENDEREÇO: (Dh/DAh)	Essa função é realizada quando se escreve no endereço de I/O Dh (DAh). Nenhum dado é associado a esse endereço. O controlador necessita de ser inicializado após um *Master Clear*.

Registrador de Estado (*Status Register)* ENDEREÇO: (8h/D0h)	O estado do controlador de DMA pode ser obtido com uma leitura no endereço de I/O 8h (D0h). Ele indica se algum canal chegou à contagem final (TC – *Terminal Count*), ou seja, se algum canal terminou a transferência. Este registrador também indica quais canais têm pedidos de DMA pendentes. A figura 8 indica a disposição dos bits deste registrador. bit 0 = fim do processo do canal 0 (TC- canal 0) bit 1 = fim do processo do canal 0 (TC- canal 1) bit 2 = fim do processo do canal 0 (TC- canal 2) bit 3 = fim do processo do canal 0 (TC- canal 3) bit 4 = sinal DREQ0 bit 5 = sinal DREQ1 bit 6 = sinal DREQ2 bit 7 = sinal DREQ3

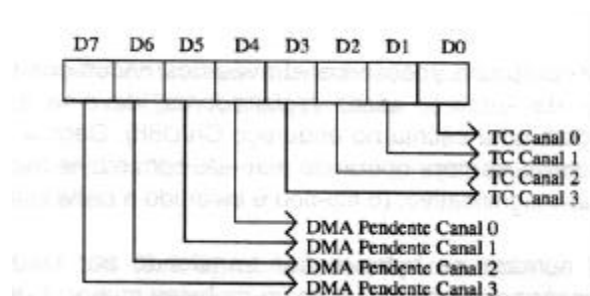



Figura 8 – Registrador de Estado

Registrador Temporário (*Temporary Register)* ENDEREÇO: (Dh/DAh)	Após uma transferência de DMA, de memória para memória, o valor do último byte transferido pode ser obtido com uma leitura no registrador temporário (endereço de I/O Dh/DAh).

2.3 – Registradores de Endereço e Contadores

Para cada canal de DMA, existem quatro registradores de dezesseis bits:

- Registrador Endereço Base.
- Registrador Endereço Corrente.
- Contador Base.
- Contador Corrente.

O endereço inicial (dezesseis bits menos significativos) apontado por cada canal é guardado no seu Registrador Endereço Base. Quando se escreve no Registrador Endereço Base, também escreve-se, automaticamente, no Registrador Endereço Corrente. Esse último aponta para a memória onde irá ser realizada a próxima transferência de DMA. O Registrador Corrente é atualizado a cada transferência. A quantidade de bytes a ser transferida é guardada no Contador Base. Quando se escreve no Contador Base, também atualiza-se, automaticamente, o Contador Corrente. O Contador Corrente indica quantos ciclos (transferências) de DMA ainda faltam. Ele é decrementado a cada transferência.

Quando um canal termina de transferir o número de bytes para que foi programado, ou seja, quando o Contador Corrente é decrementado e vai de 0 para FFFFh, o controlador de DMA gera um sinal chamado de TC (*Terminal Count*). Nesse instante, se este canal foi programado para o modo auto-inicialização, os valores do Registrador Endereço Base e Contador Base são automaticamente transferidos para o Registrador Endereço Corrente e Contador Corrente. Com isso, o canal fica pronto para repetir a operação programada.

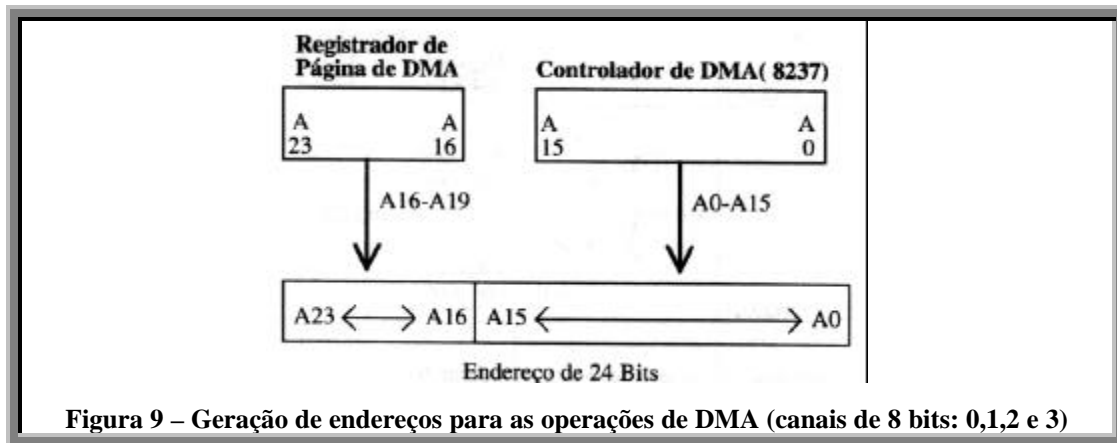
Todos esses registradores são acessados através dos endereços de I/O de 0 a 7 (C0h a CEh). Antes de atualizar esses registradores, deve-se apagar o Flip-Flop Apontador de Byte (com uma escrita no endereço Ch/D8h). Depois, pode-se escrever ou ler esses registradores, sempre operando primeiro com o byte menos significativo e depois com o byte mais significativo. (o flip-flop é invertido a cada leitura ou escrita).

Ao programar-se o número de bytes a ser transferido por DMA, deve-se tomar cuidado, pois é necessário programar o número de bytes menos 1. Isto é claro porque o sinal de fim de contagem, TC, é ativado quando o contador, ao ser decrementado, transiciona de 0 para FFFFh. Exemplos:

Número de Transferências	Valor a ser programado no contador
200	199
65536	65535 (ou FFFFh)
1	0

2.4 – Registradores de Páginas

O controlador 8237 possui registradores de endereço de 16 bits e, por isso, pode endereçar, no máximo, 64 KB. A CPU 8086/88 possui um espaço de endereçamento de 1 MB (20 bits). Já um 386 ou 486 pode endereçar até 4 GB. Parece que as operações de DMA deveriam ficar restritas aos primeiros 64 KB, o que seria uma grave limitação. Para contornar esse problema, foram colocados, à disposição dos canais de DMA, Registradores de Página, cada um com 4 bits, no XT, e 8 bits, no AT em diante. Os 16 bits do controlador de DMA, junto com os 4/8 bits do Registrador de Página, totalizam os 20/24 bits. Existe um Registrador de Página para cada canal, exceto para o Canal 0 no PC XT. O conteúdo desses registradores pode ser atualizado com um acesso de I/O. A figura 9 mostra como são reunidos o conteúdo do Registrador de Endereço (do 8237) com o conteúdo do Registrador de Página, para o caso de um canal de 8 bits (0, 1, 2 ou 3). A figura 12 aborda o caso dos registradores de página para os canais de 16 bits, que só existem a partir do 286.

Figura 9 – Geração de endereços para as operações de DMA (canais de 8 bits: 0,1,2 e 3)

Na tabela 3 estão presentes os endereços dos registradores de página associados a cada canal. Para os canais de 8 bits (0 a 3), quando ocorre um ciclo de DMA, o conteúdo do Registrador de Página é colocado nos 4/8 bits mais significativos do barramento de endereços. Assim, um único endereço de 20/24 bits é gerado para cada transferência. Isto permite que se realizem operações de DMA em qualquer página de 64 KB, dentro do espaço de 1 MB/16 MB. Ë como se todo o espaço de 1 MB/16 MB estivesse dividido em 16/256 páginas de 64KB. Esse esquema não permite que o DMA cruze a fronteira de uma página de 64KB.

Em alguns casos, necessita-se de transferir mais de 64 KB. Para isto, é necessário ultrapassar a fronteira de uma página (64 KB). O mais comum é não ter a página livre desde o início, mas sim começando em algum outro endereço dentro da página. Por exemplo, seja o caso em que se necessita transferir 90 KB. A memória está livre a partir da página 7, sendo que desta página já se usaram 20 KB. A solução seria usar os 44 KB que sobram na página 7, e ainda 46 KB da página 8. A figura 10 ilustra esse caso.

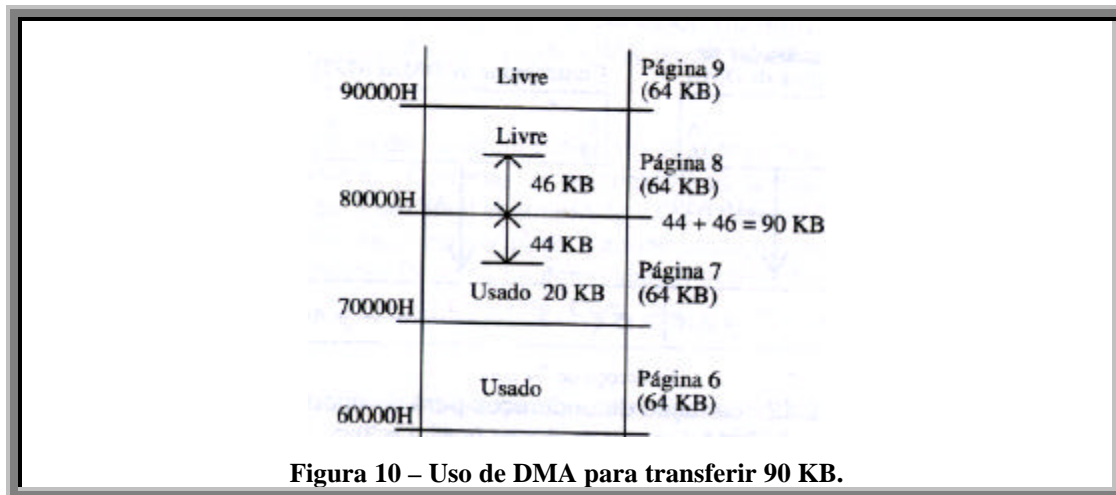
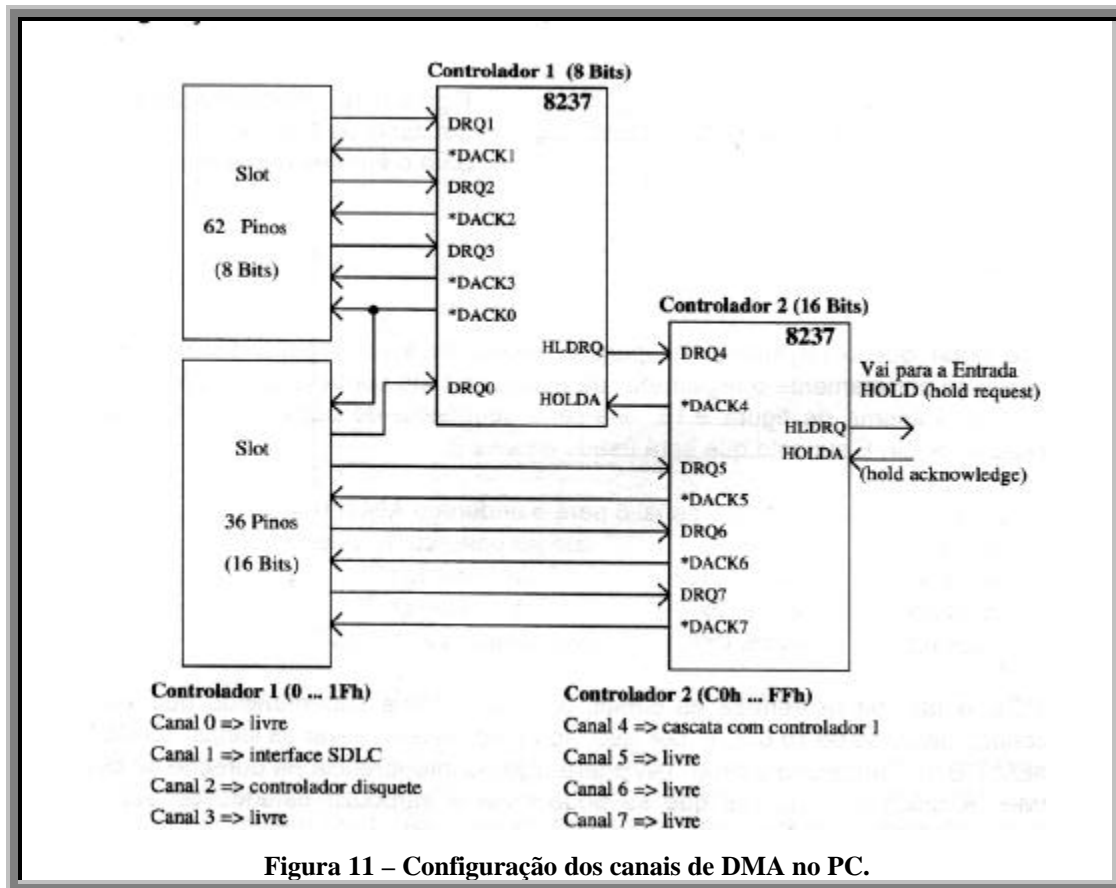



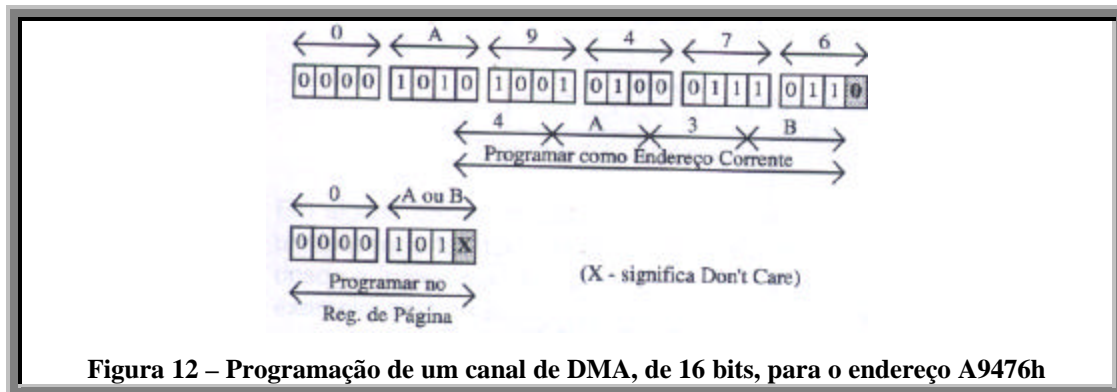
Figura 10 – Uso de DMA para transferir 90 KB.

Se o sinal TC for usado para gerar uma interrupção, pode-se obter uma excelente solução para este caso. Programa-se o Registrador de Página para a página 7 e o controlador de DMA para transferir 44 KB, a partir do primeiro endereço livre. Quando o controlador terminar a transferência, ele habilita o sinal TC, que, por sua vez, provoca uma interrupção. Esta será responsável por mudar o Registrador de Página para a página 8, e programar o Contador para transferir os 46 KB restantes. Não será necessário atualizar os Registradores de Endereço do DMA, por que eles já estarão em zero. Após transferir esses 46 KB, um novo TC será gerado e a rotina de interrupção deve parar o DMA, e sinalizar que terminou a transferência.

Os PCs 286, 386, 486 e Pentium usam para DMA um circuito equivalente ao 8237. Entretanto, o segundo controlador de DMA 8237, com quatro canais, opera no modo cascata e oferece três canais adicionais de 16 bits. O quarto canal é usado para aceitar (em cascata) a entrada do controlador original. A figura 11 ilustra a configuração dos canais de DMA do PC.

Figura 11 – Configuração dos canais de DMA no PC.

Um cuidado a ser tomado é com relação aos endereços do segundo controlador de DMA. No endereçamento deste circuito, não é usada a linha A0, e, com isso, os endereços dos diversos registradores não estão em seqüência, mas sim, saltam de dois em dois.

Os canais 5, 6 e 7 de DMA trabalham transferindo palavras de 16 bits, e, por isso, os 16 bits do endereço gerado pelo controlador de DMA, são mapeados de A1 a A16. A linha A0 não é usada porque, em uma palavra de 16 bits, o byte menos significativo é guardado no endereço par (A0=0) e o mais significativo no endereço ímpar (A0=1). No registrador de página desses canais de 16 bits, programam-se os bits de endereço de A17 até A23. A figura 12 ilustra esta programação, para o caso em que se vai realizar DMA a partir do endereço A9476h.

Figura 12 – Programação de um canal de DMA, de 16 bits, para o endereço A9476h

Deve-se notar que o registrador de página possui, mas não usa, o bit A16. Para programar-se corretamente o registrador de endereços (do controlador de DMA), para atender ao problema da figura 12, usa-se seqüência de códigos em C, que é mostrada a seguir, supondo que será usado o canal 6.

```
/* código em C para preparar o canal 6 para o endereço A9476h*/
        outportb    (0xD8,0);      /* apagar ponteiro de bytes */
        outportb    (0xC8,0x3B);   /* LSB do endereço */
        outportb    (0xC8,0x4A);   /* MSB do endereço */
        outportb    (0x89,0xA);    /* Registrador de Página */
```

Nos PCs, o fato de usarem-se os canais 5, 6 ou 7 torna subentendido que serão transferidas palavras de 16 bits, e, por isso, não é necessário ativar as linhas IO CS16 ou MEM CS16. Tampouco a linha OWS terá alguma interferência na duração do ciclo de DMA. A única interferência que se pode fazer é introduzir estados de espera, através da linha IO CH RDY.

Existe uma diferença significativa no modo de operação dos canais de DMA do AT. Os canais normalmente não operam com o mesmo relógio da CPU; mas é comum usarem-se ½, 1/3 ou ¼ desta freqüência. Além disso, as CPUs 286, 386 e 486 necessitam de mais períodos de relógio para acessar, via endereçamento de I/O, os controladores de DMA. A figura 13 ilustra um pequeno circuito para transferir por DMA, palavras de 16 bits.

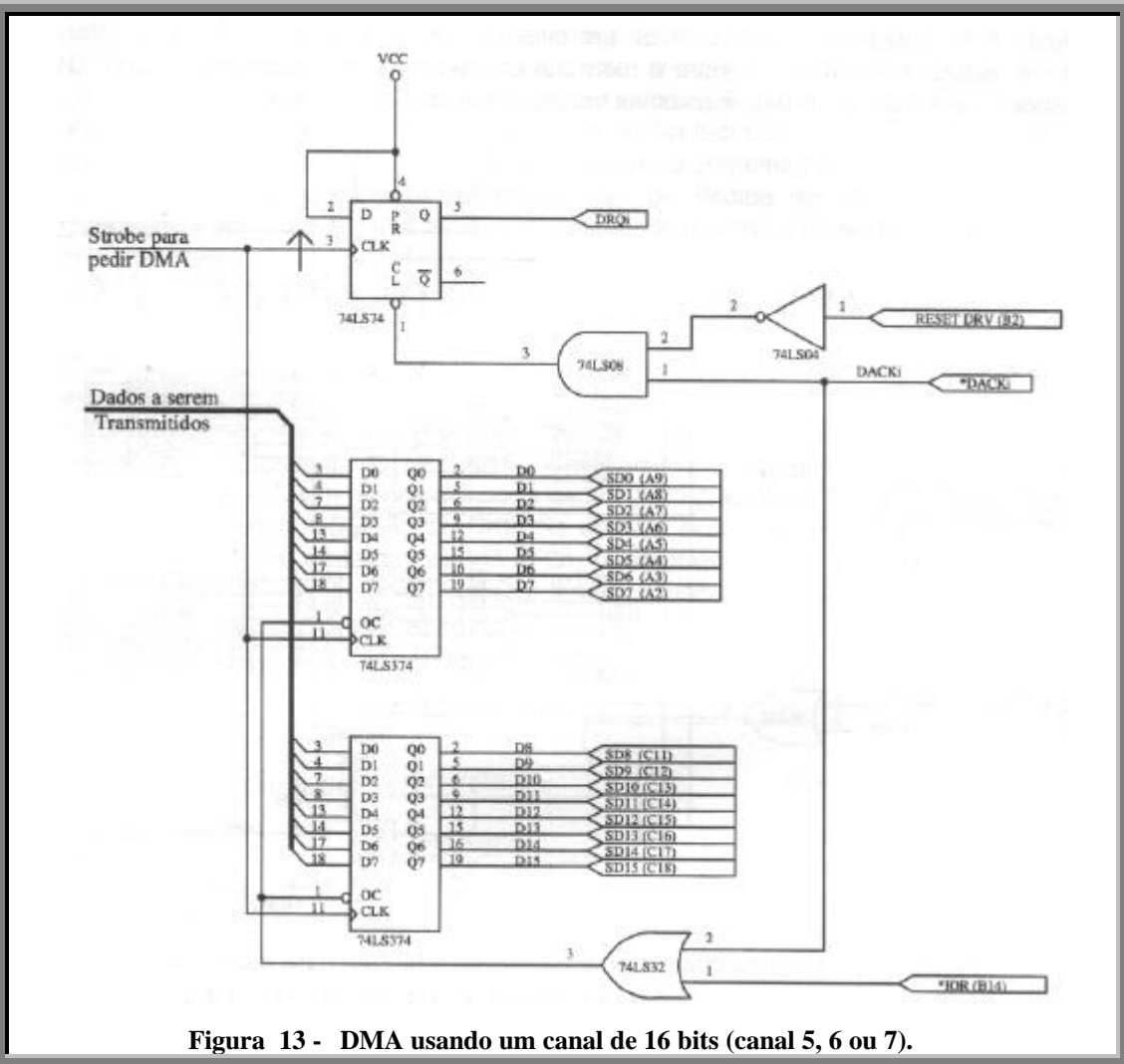
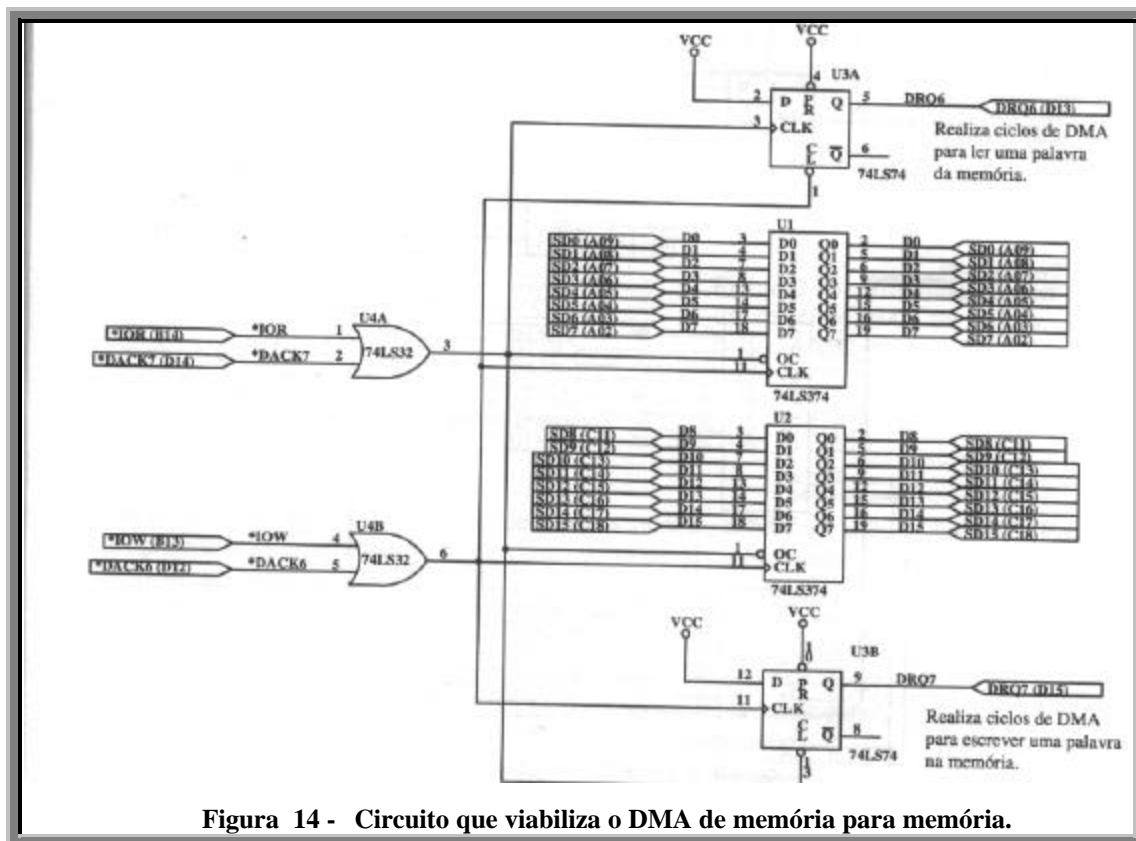



Figura 13 - DMA usando um canal de 16 bits (canal 5, 6 ou 7).

Com o uso dos canis de DMA oferecidos no PC, é possível realizar transferências de dados entre a memória convencional e a estendida. No PC, os registradores de página de DMA aumentam em 8 bits o endereço gerado pelo controlador, resultando em um total de 24 bits de endereço. Isto permite acessar os primeiros 16 MB de memória. O esquema de memória usado nos PCs normalmente não permite operações de DMA de "memória para memória", mas é possível adicionar um circuito que permita a simulação deste modo. Com tal circuito, transferem-se dados entre a memória convencional e a estendida, sem a necessidade de um gerenciador de memória que faz transições momentâneas para o modo protegido. No entanto, deve-se tomar cuidado para não invadir áreas do sistema. Uma outra vantagem é a velocidade da transferência, que deverá ser mais rápida que qualquer outra obtida por software. A figura 14 apresenta a sugestão de um circuito que, usando dois canais de DMA, pode

realizar transferências entre a memória convencional e a estendida. Como são usados os canais de 16 bits, é possível transferir um total de 128 KB.

Figura 14 - Circuito que viabiliza o DMA de memória para memória.

O funcionamento elétrico deste circuito é muito simples. Inicialmente, deve ser notado que os *latches* U1 e U2 trabalham em paralelo e permitem o manejo de palavras de 16 bits. O canal 6 deve ser programado para realizar ciclos de leitura, ou seja, ler da memória e guardar no *latch*, enquanto que o canal 7 deve ser programado para realizar ciclos de escrita, ou seja, escrever o conteúdo do *latch* na memória. Supondo que os canais estejam corretamente programados, e que o pedido DRQ 6 esteja ativo, é iniciado um ciclo de DMA, que irá ler uma palavra da memória. Esta palavra será escrita no *latch* quando os sinais IOW e DACK6 forem ativados (em nível baixo, através do U4A). Neste instante, o pedido DRQ6 será removido, através da entrada CL do flip-flop U3A. Porém, o mesmo sinal que apaga o pedido DRQ6 ativa o pedido DRQ7, através da entrada CLK do flip-flop U3B. Agora que DRQ7 está ativado, o sistema realiza um ciclo de DMA, que escreve o conteúdo do *latch* na memória. Notar

que o *latch* coloca a palavra no barramento quando os sinais IOW e DACK7 são ativados (em nível baixo, através de U4B). A ativação desses dois sinais apaga, através da entrada CL, o pedido DRQ7, mas, ao mesmo tempo, ativa o pedido DRQ6, através da entrada CLK do flip-flop U3A. Agora, o ciclo se repete até chegar à contagem final (TC). Falta conseguir o primeiro pedido DRQ6, que pode ser facilmente gerado através do Registrador de Pedido do 8237. É conveniente colocar uma porta auxiliar que zere os dois flip-flops quando o Reset for ativado.

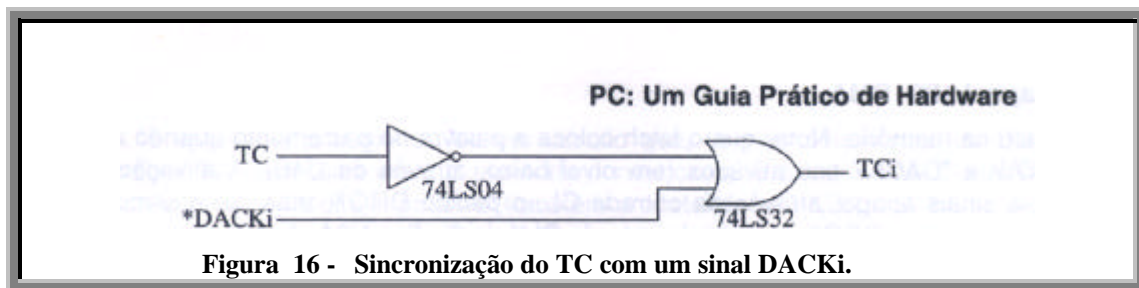
2.5 – Reutilização de um Canal de DMA

O uso de um canal de DMA por uma interface não impede que o mesmo seja utilizado em um outro projeto. É óbvio que as duas extremidades não podem usar simultaneamente o mesmo canal, ma, se não é necessário simultaneidade, duas interfaces podem compartilhar o mesmo canal de DMA. Um exemplo disto é o projeto do Adaptador de Disquete do PC. Um bit de uma porta de I/O é usada para habilitar e desabilitar os sinais DRQ2 e DACK2. Se um 0 é escrito no bit 3 do Registrador de Saída, que está no endereço de I/O 3f2h, o adaptador se desconectará das linhas DRQ2 e DACK2, deixando-as livres para serem usadas por um outro adaptador. A figura 15 ilustra como isso pode ser feito para um circuito qualquer.

Figura 15 - Circuito para permitir o compartilhamento de um canal de DMA

2.6 – Sinal de Fim de Contagem (TC – Terminal Count)

O controlador 8237 gera um sinal, denominado TC, cada vez que um dos quatro canais chega à condição de fim de contagem, ou seja, quando o controlador corrente está em zero e realiza-se mais uma transferência de DMA. Normalmente, este sinal é usado pela interface para terminar as transferências. Se está sendo usado o modo de auto-inicialização, ele serve para indicar à interface que se deve iniciar um novo conjunto de transferências. Uma dificuldade é que o controlador gera um único sinal TC para os 7 canais. Para saber a que canal corresponde o TC gerado, é necessário fazer um AND com o DACK. A figura 16 ilustra o circuito.

Figura 16 - Sincronização do TC com um sinal DACKi.

3 - INTERRUPÇÕES

O uso de interrupções é útil e freqüentemente necessário no projeto de interfaces com computadores. Sua maior vantagem está na capacidade de fazer o periférico receber a imediata atenção da CPU, sem que seja necessário um "*polling*" (técnica onde a CPU consulta o periférico se há a disponibilidade do dado). Isso deixa o processador livre para fazer outras coisas enquanto sua atenção não é solicitada pela interface. Um bom exemplo de uso da interrupção é o serviço do teclado do PC. Cada tecla, ao ser acionada, gera pelo menos uma interrupção. Com isso, a CPU fica livre para processar, e atende ao teclado somente quando ele solicita. A interrupção é freqüentemente usada para interfacear aplicações que requeiram sincronização com eventos externos, ou quando surge uma situação de erro que requer a atenção imediata da CPU.

3.1 – Interrupções no PC

As CPUs da família 80x86 oferecem duas entradas para interrupção: uma entrada mascarável (INTR) e uma entrada não mascarável (NMI). O termo mascarável apenas significa que a interrupção pode ser desabilitada por software. Na arquitetura do PC, a NMI assume um papel importante e é usada para alertar sobre situações catastróficas, ou de extrema importância. Existem três possíveis causas para a interrupção não mascarável:

- erro de paridade na memória do sistema;
- erro de coprocessador e;
- erro no canal de I/O.

Como a lógica interna das CPUs não permite a desabilitação por software da NMI, na placa do PC foi colocado um bit (em hardware) de I/O que impede a chegada dessa interrupção até a CPU. Um controlador de interrupções é utilizado para gerenciar a interrupção mascarável (INTR), expandindo-a para oito ou dezesseis entradas (com prioridades programáveis). Algumas interrupções são usadas internamente na placa do sistema, enquanto que outras estão disponíveis no barramento de expansão. A figura

17 apresenta um esquema completo com as dezesseis interrupções do PC (NMI, IRQ0, IRQ1, IRQ3 a IRQ15). O circuito controlador de interrupções é compatível com o 8259 da seguinte forma. No PC XT, utilizava-se um 8259. Do PC AT em diante, usam-se dois 8259 cascateados.

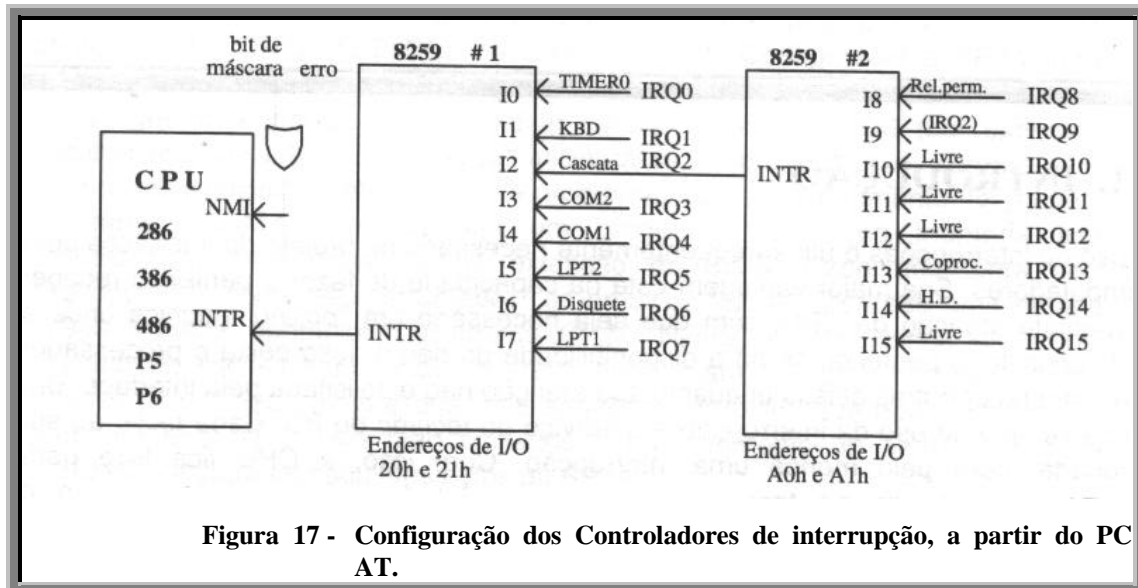



Figura 17 - Configuração dos Controladores de interrupção, a partir do PC AT.

No 286, 386, 486 e Pentium, a IRQ2 não está mais disponível no barramento, porque ela serve para o cascateamento dos dois controladores, tendo sido substituída pela IRQ9, do segundo controlador. Para manter a compatibilidade com dispositivos antigos que usavam a IRQ2, por software, redirecionam-se as interrupções geradas pela IRQ9 para as rotinas de serviço da IRQ2.

3.2 – O Controlador de Interrupções

O controlador de interrupções 8259 expande a interrupção mascarável da CPU em até 8 entradas com prioridade. A figura 18 apresenta um diagrama de blocos deste controlador . Ele recebe os pedidos de interrupção em um banco de 8 latches, chamado Registrador de Pedidos de Interrupção (“*Interrupt Request Register*” - IRR). O IRR pode ser programado para programar por flanco (↑) ou por nível. Um Registrador de Máscara de Interrupção (“*Interrupt Mask Register*” – IMR) é programado de forma a habilitar ou desabilitar os pedidos de interrupção.

Depois do pedido de interrupção ser ativado no IRR, ele vai para o Resolutor de Prioridades. O resultado da decisão da prioridade é enviado para um outro registrador de 8 bits, chamado de Registrador em Serviço (*"In Service Register"* – ISR). Esse registrador reflete o nível de interrupção que está atualmente em serviço para a CPU. O controlador 8259 pode ser programada para atender a diversos modos de prioridade. Além disso, ele permite mascarar ou apagar um pedido de interrupção, através da programação de registradores de controle. Durante a operação do controlador, é possível ler o estado de diversos registradores internos.

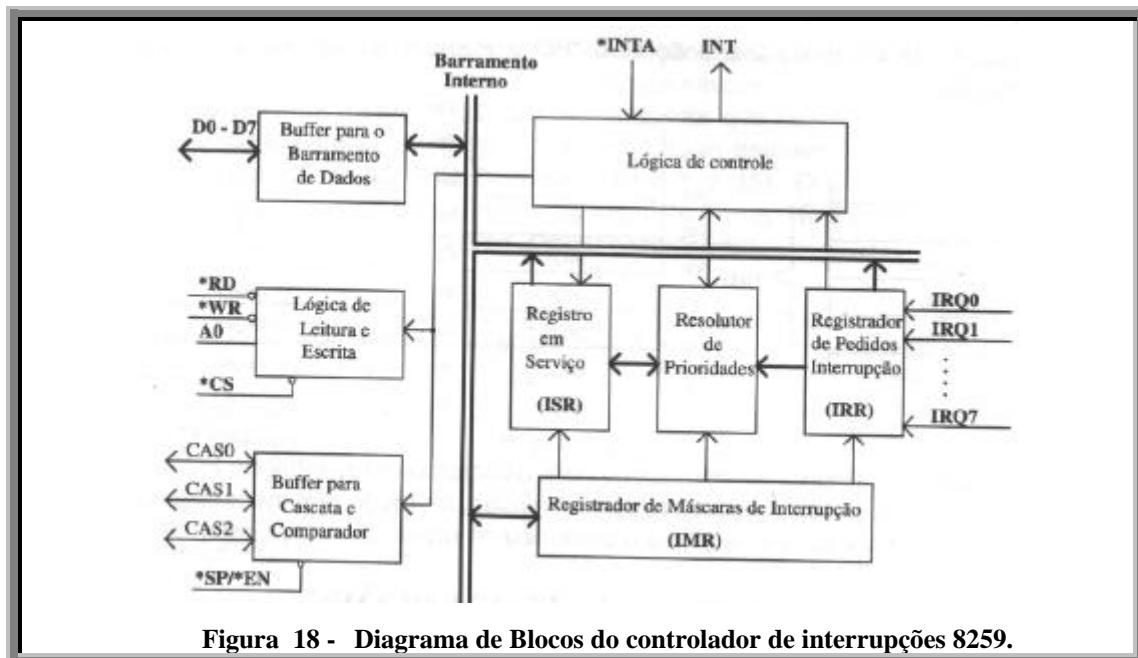



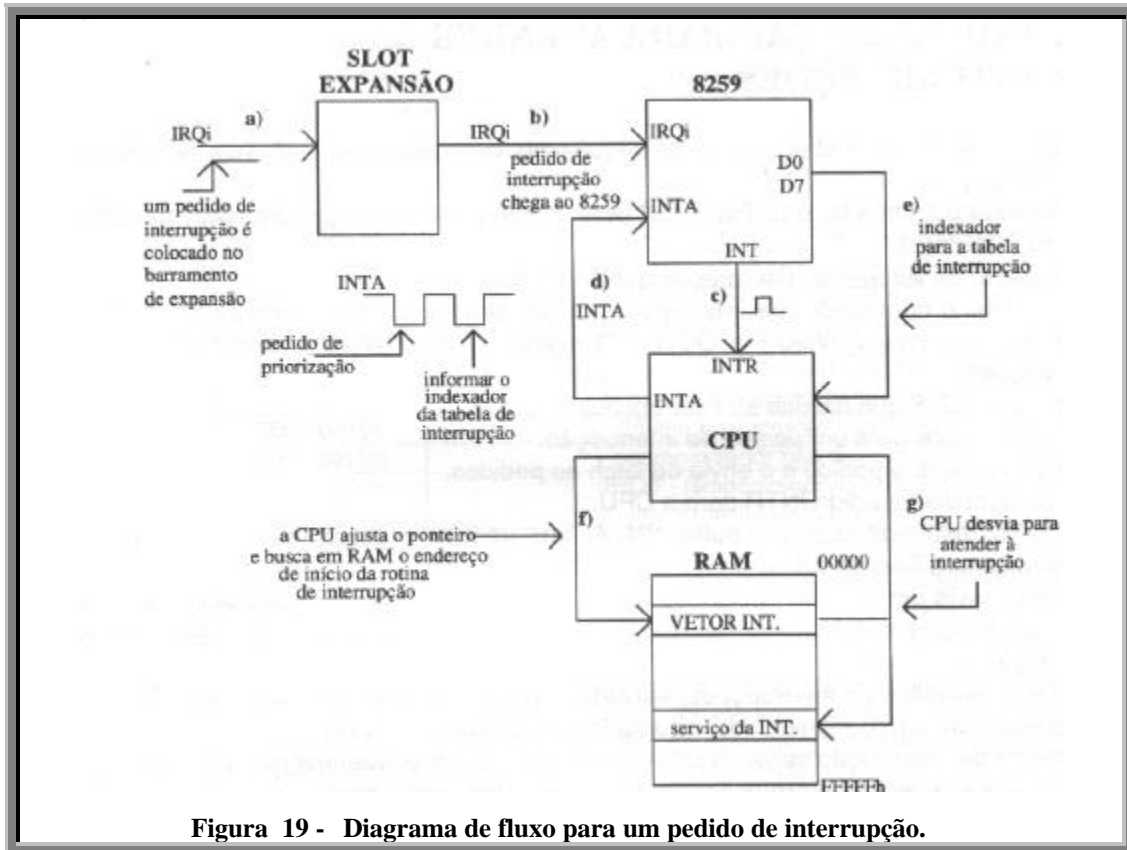
Figura 18 - Diagrama de Blocos do controlador de interrupções 8259.

No caso do PC 286, 386, 486, Pentium (e também para o PS/2), usam-se dois 8259 em cascata, o que tornam possíveis 15 entradas de interrupção. Não é possível fazer outros cascateamentos de 8259, pois as linhas que permitem esse modo não estão disponíveis no barramento de expansão.

3.3 – Sequência de Eventos em uma Interrupção

Quando um pedido de interrupção chega ao sistema, uma seqüência de eventos transforma esse pedido em chamada de subrotina que irá atendê-lo. A figura 19 apresenta um diagrama de fluxo ilustrando os eventos que ocorrem na aceitação de uma interrupção. Para que tudo isso seja possível, é necessária uma fase de inicialização. Supondo que essa fase já esteja pronta, e que a interrupção não esteja mascarada, os seguintes eventos acontecem:

1. Uma interface ativa ($\uparrow$) uma linha de pedido (IRQ) de interrupção.
2. O controlador de interrupção (8259) recebe o pedido e resolve a prioridade deste pedido com os outros pedidos pendentes ou que estão chegando.
3. Se o pedido é o único ou se é o de mais alta prioridade entre os pendentes, ao terminar a rotina de prioridade superior, um pedido de interrupção (INT) é enviado à CPU.
4. A CPU responde com dois pulsos INTA (para o 8259). O primeiro INTA congela as prioridades e escreve no Registrador em Serviço (ISR) o nível da interrupção de mais alta prioridade. O segundo INTA indica que o 8259 deve colocar, no barramento de dados, um ponteiro de 8 bits.
5. A CPU recebe o ponteiro de 8 bits e usa-o para indexar a tabela com os vetores de interrupção. No modo real, em cada posição dessa tabela, existem os valores do seletor de segmento (CS) e do offset (IP) que especificam onde está a rotina de serviço (manipulador) para a interrupção recém aceita. No modo protegido, a tabela de vetores contém descritores que definem a interrupção.
6. A CPU guarda na pilha os valores atuais de IP, CS e flags, quando no modo real (EIP, CS e EFLAGS, no modo protegido), carrega os novos valores de CS e IP (EIP, no modo protegido), a partir da tabela de vetores de interrupção.
7. Executa-se o serviço do manipulador de interrupção.

Figura 19 - Diagrama de fluxo para um pedido de interrupção.

É importante ressaltar que antes de uma rotina de serviço de interrupção ser executada, alguns cuidados devem ser tomados. Primeiro, a rotina deve guardar na pilha o conteúdo dos registradores que serão usados. Isso permite que se restaure os registradores ao terminar a rotina. Segundo, deve-se apagar o pedido de interrupção e resetar o latch em serviço através do comando EOI (fim de interrupção) para o 8259 (*outportb* (0x20, 0x20) em C). Isso permite que o 8259 receba novas interrupções.

Pedidos adicionais de interrupção, por parte do 8259, são mascarados pela CPU enquanto a rotina de interrupção está sendo executada. É possível permitir novas interrupções ao mudar o bit de habilitação de interrupção (IF) que está no registrador de flags. Assim, uma interrupção de prioridade mais alta pode interromper a atual. Ao final da rotina de interrupção, deve-se restaurar os registradores que foram guardados na pilha. Os flags são automaticamente restaurados com uma instrução IRET, que ordena o regresso da interrupção.

3.4 – Inicialização para Atender às Interrupções

A seguir, abordam-se as seqüências de interrupção, incluindo as etapas de inicialização:

1. Inicializar o apontador de pilha (E)SP para a área onde serão guardados os estados e os registradores.
2. Inicializar os vetores de interrupção que serão usados.
3. Inicializar o controlador de interrupções 8259, através de seus registradores ICW (“*Interrupt Command Word*”) e OCW (“*Operation Control Word*”). Desmascarar as interrupções.
4. Ativar a flag IF que habilitação as interrupções.
5. Uma interface gera um pedido de interrupção.
6. O 8259 recebe o pedido e o envia ao latch de pedidos.
7. A CPU responde com um pulso INTA, que prioriza o pedido e o envia para o Registrador em Serviço (ISR), dentro do 8259.
8. A CPU envia um segundo pulso de INTA . O 8259 responde a esse pulso com o um valor que indica o nível de interrupção. Esse valor é usado pela CPU para associar a interrupção a um vetor.
9. A CPU desabilita as interrupções, salvando na pilha o registrador de flags, (E)IP e CS, desviando para o manipulador apontado pela tabela de vetores.
10. A rotina de interrupção salva na pilha o conteúdo dos registradores que vai usar.
11. Para permitir novas interrupções, a rotina de interrupção envia um comando EOI (“*End Of Interrupt*”) para o 8259.
12. A rotina de interrupção é executada.
13. Ao completar a rotina, restauram-se os conteúdos dos registradores usados.
14. A rotina de interrupção executa uma instrução de retorno de interrupção, restaurando os valores IP, CS e flags (isto habilita interrupções, pois carrega IF) (EIP, CS e EFLAGS, no modo protegido),regressando do manipulador, e continuando a execução do programa interrompido.

3.5 – Vetores de Interrupção

No modo real, cada endereço é formado por duas palavras de 16-bits: o Ponteiro de Instrução (IP) e o Segmento de Código (CS), e a tabela de vetores de interrupção é constituída pelos primeiros 1024 bytes da memória física. Quando ocorre uma interrupção, a tabela de vetores é acessada para obterem-se os novos valores de IP e CS, que especificam o endereço para onde será desviada a execução. Convém lembrar que as interrupções podem ser geradas de diversas formas: internamente (exceções pré-definidas), por software, e externamente (por hardware do usuário). Já no modo protegido, o endereço do manipulador de interrupções é obtido indiretamente, via descritores da Tabela de Descritores de Interrupção. A figura 20 relaciona alguns vetores de interrupção para o modo real do PC.

Vetor	Ponteiro no modo real	Descrição
00h	0000:0000h	divisão por zero
01h	0000:0004h	*single step*
02h	0000:0008h	interrupção não mascarável
03h	0000:000Ch	*breakpoint*
04h	0000:0010h	*overflow*
05h	0000:0014h	tecla *print screen* - envia conteúdo da tela à impressora ou, exceção de erro de fronteira
06h	0000:0018h	código ou instrução inválidos
07h	0000:001Ch	coprocessador não disponível
08h	0000:0020h	IRQ0 - temporizador do sistema ou, exceção de dupla falta
09h	0000:0024h	IRQ1 - teclado ou, exceção de sobreposição de segmento
0Ah	0000:0028h	IRQ2 - cascateamento do 8259 ou, exceção de TSS inválido
0Bh	0000:002Ch	IRQ3 - com2/com4 ou, exceção de segmento ausente
0Ch	0000:0030h	IRQ4 - com1/com3 ou, exceção de erro de pilha
0Dh	0000:0034h	IRQ5 ou, exceção de falta geral na proteção
0Eh	0000:0038h	IRQ6 - *flopy disk-drive* ou, exceção de falta na paginação
0Fh	0000:003Ch	IRQ7 - impressora
10h	0000:0040h	serviços de vídeo ou, erro de coprocessador
11h	0000:0044h	serviços para determinação do equipamento
12h	0000:0048h	serviços para obtenção do tamanho da memória
13h	0000:004Ch	serviços de disco
14h	0000:0050h	serviços para comunicação serial
15h	0000:0054h	serviços do sistema
16h	0000:0058h	serviços do teclado

Figura 20 - Relação de alguns vetores de interrupção para o modo real do PC.

3.6 – Inicialização do Controlador de Interrupções

O primeiro controlador de interrupções é acessado através dos endereços de I/O 20h e 21h, e o segundo ocupa os endereços A0h e A1h. Cada controlador possui 7 registradores, o que deixa claro que existem diversos registradores compartilhando o mesmo endereço. O 8259 faz este compartilhamento ao usar o primeiro endereço para o modo de incialização (feita uma seqüência de escritas) e, então, usando o mesmo endereço para o modo de operação, que é iniciado automaticamente após o modo de inicialização ser completado. A tabela 3 apresenta o endereço dos diversos registradores. O 8259 possui dois modos:

- Incialização: escrevem-se as palavras ICW1, ICW2, ICW3 e ICW (ICW - "*Inicialization Command Word*").
- Operação: escrevem-se as palvras OCW1, OCW2 e OCW3 (OCW – "*Operation Command Word"*)

Tabela 3 - Endereços dos registradores do 8259.

Primeiro 8259	Segundo 8259	Registrador	Acesso a qualquer instante
20h	A0h	ICW1	sim
21h	A1h	ICW2	não
21h	A1h	ICW3	não
21h	A1h	ICW4	não
21h	A1h	OCW1	sim
20h	A0h	OCW2	sim
20h	A0h	OCW3	sim

Entra-se no modo de inicialização a qualquer instante, basta escrever-se no endereço 20h (A0h) um byte com o bit 4 igual a 1. O 8259 usa este byte como ICW1. Até 3 palavras de incialização podem ser enviadas após ICW1, elas usam o endereço 21h (A1h). Essas palvras devem ser enviadas na ordem mostrada na figura 21. Após o envio da última ICW, o 8259 entra no modo de operação. Uma escrita no endereço 21h (A1h) especifica a palvra OCW1. A palavra OCW2 é escrita no endereço 20h

(A0h), com os bits 3 e 4 iguais a zero. A palavra OCW3 é escrita no endereço 20h (A0h), com bit 3 igual a 1 e obit 4 iagual a zero. A tabela 4 mostra um resumo das escritas no endereço 20h (A0h).

Figura 21 - Sequência para a inicialização do controlador 8259.

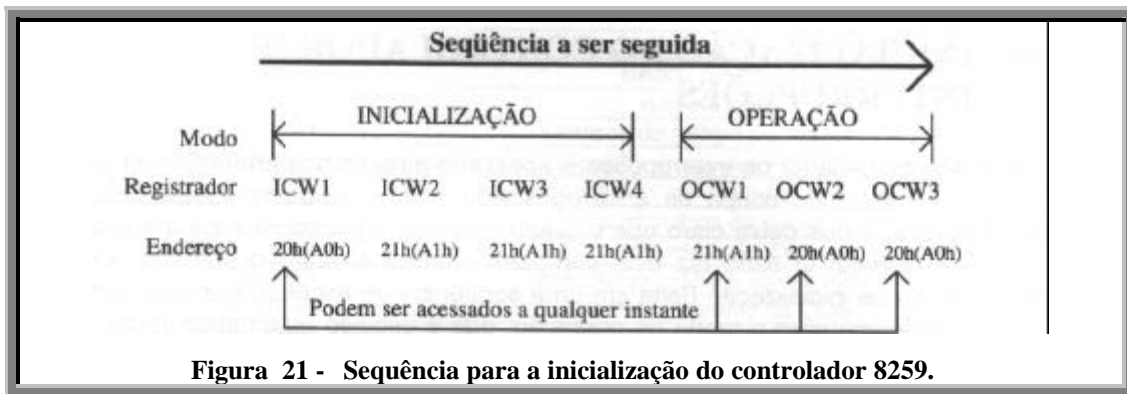



Tabela 4 – Registradores do 8259 que compartilham o endereço 20h (8259 mestre).

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Operação
X	X	X	1	X	X	X	X	ICW1
X	X	X	0	0	X	X	X	OCW2
X	X	X	0	1	X	X	X	OCW3

A seguir, é apresentado um resumo.

a) Modo de Inicialização:

- Com a escrita de um dado, com bit 4 igual a 1, no endereço 21h (A1h), coloca-se o 8259 no Modo de Inicialização. Esse dado é interpretado como ICW1.
- Os dados ICW2, ICW3, ICW4 e OCW1 devem ser enviados em seqüência, usando-se o endereço 21h (A1h). Com o envio de ICW4, o 8259 entra, automaticamente, no modo de operação e recebe o OCW1.

b) Modo de Operação:

- Normalmente, o 8259 está neste modo. Escrevem-se OCW2 e OCW3, conforme mostrado na figura 22.

c) Palavras de Comando para a Inicialização (ICW)

Quando a ICW1 é escrita no 8259, ocorre a seguinte seqüência de inicialização:

a) O modo sensível por flanco é desativado.
b) A máscara de interrupções é zerada (todas ativadas).
c) A IRQ 7 (IRQ 15) recebe a menor prioridade.
d) Se o bit IC4=0, então todas as funções selecionáveis através de ICW4 são zeradas.

Tabela 5 – Formato da palavra ICW1.

BIT	DESCRIÇÃO
D0(IC4)	indica se a seqüência de inicialização vai incluir, ou não, a palavra ICW4.
D1(SNGL)	indica se existe mais de um 8259 presente.
D2(ADI)	indica se existe mais de um 8259 presente.
D3(LTIM)	indica se a interrupção será ativada por nível ou flanco.
D4	deve ser 1.
D5, D6, D7	não são usados no modo 8088.

Tabela 6 – Formato da palavra ICW2.

BIT	DESCRIÇÃO
D0 – D2	não são usados no modo 8088.
D3 – D7	são os 5 bits mais significativos do índice (8 bits) enviado com o segundo INTA.

Tabela 7 – Formato da palavra ICW3.

BIT	DESCRIÇÃO
D0 – D7	não são usados no PC XT, porque sua arquitetura não oferece recursos paar um segundo 8259, mas são usados no PC AT. Na inicialização, esta palavra é saltada pois em ICW1 é especificado o “modo sozinho” (SNGL=1). Assim, a terceira palavra de inicialização que se escreve é ICW4.

Tabela 8 – Formato da palavra ICW4.

BIT	DESCRIÇÃO
D0(PM)	indica se o 8259 deve trabalhar no modo 8085 ou no modo 8088.
D1(AEIO)	este bit permite o *reset* automático do pedido de interrupção, no registrador em serviço (ISR), tão logo a CPU aceite a interrupção. Isto acontece com a chegada do segundo pulso de INTA. Assim que o bit do ISR é apagado, o controlador está habilitado para enviar à CPU o pedido de uma interrupção de nível mais alto. Isto pode ser desejado ou não, de acordo com a interrupção que está em andamento. No caso do PC, esse bit é zerado, o que indica que a rotina de interrupção deve enviar um comando EOI para apagar o bit do ISR e permitir outras interrupções. Em linguagem C, esse comando é realizado com "*outportb*(0x20, 0x20)".
D2(M/S)	indica se o 8259 é mestre ou escravo.
D3(BUP)	indica ao controlador que ele está em um sistema com barramento de dados amplificados, de forma que ele deve gerar um sinal de controle, para habilitar o buffer do barramento de dados durante as interrupções.
D4(SFNM)	indica ao controlador que ele está em um sistema com diversos controladores, e que deve usar o modo de aninhamento completo para estabelecer a prioridade entre os controladores.
D5, D6, D7	não são usados.

Agora, o controlador 8259 está inicializado e pronto para aceitar pedidos de interrupção. Durante a operação do controlador, pode ser necessário mudar o modo de operação ou ler o estado das interrupções. Para habilitar essas funções, o controlador vai para o modo de operação assim que termina a inicialização. Nesse modo, ele aceita as palavras de controle da operação (OCW).

d) Palavras de Controle da Operação (OCW)

No modo operação, a interrupção muda e as escritas acontecem nas palavras de controle da operação (OCW). Existem três OCW:

- OCW1: endereço I/O 21h (A1h) (figura 8.13)
- OCW2: endereço I/O 20h (A0h), com D4=0 e D3=0 (figura 8.14)
- OCW3: endereço I/O 20h (A0h), com D4=0 e D3=1 (figura 8.15)

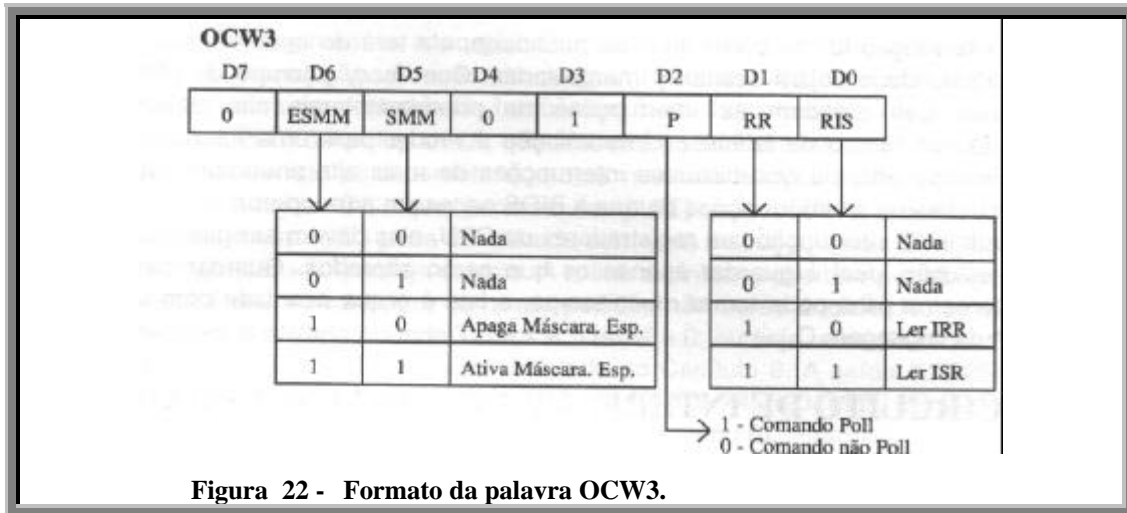
Os bits de OCW1 são usados para ativar ou desativar as interrupções, e, juntos, constituem a máscara de interrupção. O bit de máscara, quando ativado (1), desabilita a interrupção correspondente. Quando o bit de máscara está desativado (0), a interrupção está habilitada. O valor atual desse registro pode ser obtido com uma leitura no endereço de I/O 21h (A1h).

Tabela 9 – Formato da palavra OCW1.

BIT	DESCRIÇÃO
D0 – D1	indicam a interrupção de mais baixa prioridade. Assim, se a prioridade mais baixa é a IRQ 5, a IRQ 6 será a de mais alta prioridade (6, 7, 0, 1, 2, 3, 4, 5).
D3 e D4	devem estar zerados.
D5 – D7	a combinação destes 3 bits especificam diversos comandos.
(001)	Com esse valor em D5-D7, executa-se o comando EOI não específico. Se o modo EOI automático (em ICW4) não foi ativado, como é o caso do PC, o pedido de interrupção deve ser apagado com este comando EOI. Ë exatamente isto que faz o comando "*outportb (0x20, 0x20)*", ou as instruções mov ax,20h e out 20h, ax.

Tabela 10 – Formato da palavra OCW2.

BIT	DESCRIÇÃO
D0 – D1 **(RIS –RR)**	esses bits permitem selecionar o "modo de leitura de esatdo". Valores 00 e 01 não causam qualquer ação. O valor 2 (10) permite ler o valor do Registrador de Pedidos de Interrupção (IRR) na próxima operação de leitura do controlador. O valor 3 (11) permite ler o Registrador de Interrupção em Serviço (ISR) na próxima operação de leitura. Esta próxima leitura pode ser feita no endereço 20h (A0h) ou 21h (A1h).
D2 (P)	envia um comando ao controlador que realiza a mesma função que o primeiro INTA, ou seja, congela o pedido de interrupção de mais alta prioridade no ISR. Na próxima, em D0-D2, estará a interrupção que foi aceita. Não é usada pela BIOS.
D3 – D4	a combinação destes 3 bits especificam diversos comandos.
D5 – D6 **(SMM e ESMM)**	permite um uso especial do registrador de máscara. Não é usado no PC.

Figura 22 - Formato da palavra OCW3.

Como pode ser visto, pela quantidade de modos de inicialização e operação, o controlador 8259 é um dispositivo complexo que realiza um importante controle sobre a operação do PC. SE não for tomado o cuidado devido, uma mudança nesses modos pode ter um forte impacto sobre a operação do PC e alguns softwares talvez não funcionem corretamente. Assim, é recomendado que os aplicativos usem os modos de inicialização e operação programados pela BIOS.

3.6 – Desempenho da Interrupção

O tempo gasto, desde o instante em que o pedido de interrupção é colocado no barramento até o instante em que é executada a primeiro instrução da rotina de interrupção, é chamado Tempo de Latência da Interrupção. Em muitas aplicações, esse tempo pode ser crítico. Devido às variações envolvidas, é muito difícil especificar o tempo de latência geral. Cada caso terá de ser estudado em particular. Os principais itens envolvidos são:

1. Tempo de processamento do hardware: é o tempo que a CPU gasta a partir da chegada do pedido do 8259. Estão envolvidos os tempos para:
    - receber o ponteiro do 8259;
    - guarda na pilha os flags e o endereço atual;
    - desviar para a rotina de interrupção.

2. As interrupções somente podem ser aceitas ao final de uma instrução. Assim, se um pedido de interrupção chega durante o início de uma instrução, ele terá de aguardar que a CPU termine a execução da instrução em curso. As instruções de multiplicação e divisão gastam muito tempo e necessitam de uma análise mais cuidadosa. Os prefixos de "*repeat, lock*" e "*segment override*" são considerados como parte da instrução que modificam, e, então nenhuma interrupção pode ocorrer entre o prefixo e a instrução. As operações de MOV ou POP, em um registrador de segmento, impedem o reconhecimento da interrupção até a execução da próxima instrução.

3. Se a interrupção for de baixo nível de prioridade, ela terá de aguardar que as de mais alta prioridade sejam aceitas e manipuladas. Com isso, o tempo de execução das rotinas que atendem às interrupções de prioridade mais alta devem ser adicionados ao tempo de latência. Uma solução é mudar para uma interrupção de prioridade mais alta, ou desabilitar as interrupções de mais alta prioridade.

4. Se a rotina de interrupção usa registradores da CPU, eles devem ser guardados na pilha. A solução ideal é guardar apenas os que serão alterados. Guardar todos os registradores na pilha pode tomar muito tempo, e isto é o que acontece com o "void interrupt" da linguagem C.

4 - HARDWARE

Para o desenvolvimento do trabalho foi utilizado um microcontralador PIC16C73A, onde foram programadas as funções de DMA e IRQ. O acesso ao barramento do microcomputador é feito através de uma placa "buffer", composta por 4 circuitos integrados 74LS245 (*Buffer octal tri-state*) e um conector para *flat cable* de 50 pinos. Os sinais do barramento do PC disponíveis no conector são:

A0 – A19 (endereços) saída:	Estes são os bits usados para endereçar a memória do sistema e os dispositivos de I/O. Eles são gerados pela CPU durante acessos à memória e aos dispositivos de I/O. Durante os ciclos de DMA, eles são gerados pelo controlador de DMA. Com 20 linhas de endereço, pode-se acessar até 1 MB de memória. A CPU, através de instruções de IN e OUT, pode acessar até 64 KB dispositivos de I/O, pois essas instruções emitem endereços pelas linhas A0 até A15. As linhas A16 até A17 permanecem desativadas durante as operações de I/O. No buffer implementado estão disponíveis os bits de A0 a A9.
D0 – D7, (dados) bidirecional:	Estas linhas são usadas para transmitir dados entre a CPU e a memória ou dispositivos de I/O. Elas estarão válidas um pouco antes do flanco ascendente (↑) dos sinais (S) MEMW e IOW. Normalmente, usam-se o IOW como *strobe* para escrita nos dispositivos de I/O e a linha (S) MEMW como *strobe* de escrita na memória. Durante as operações de leitura (IOR ou (S) MEMR), os dados devem estar válidos um pouco antes do flanco ascendente (↑) de IOR ou (S) MEMR. Nas operações de DMA, essas linhas permitem que o dado trafegue diretamente entre a memória e o dispositivo de I/O. No buffer implementado estão disponíveis todos esses bits.
D8- 15, (dados) bidirecionais:	o barramento de dados do sistema (bits 8 – 15) é constituído por oito linhas bidirecionais e permite, junto com D0 – D7, realizar transferência de 16 bits. Se as interfaces possuem barramento de 8 bits, devem conectar–se apenas com o *slot* de 62 pinos. Quando uma interface desse tipo necessita transferir dados de 16 bits, a placa do sistema força dois ciclos de barramento (usando D0 – D7). As interfaces devem indicar sua capacidade para transferir dados de 16 bits, através da ativação dos sinais MEM CS16 (para dispositivos de memória) e I/O CS16 (para dispositivos de I/O). No buffer implementado estão disponíveis todos esses bits.

AEN (*Address Enable*), saída: é gerado pela lógica de controle de DMA, e serve para indicar que se está executando um ciclo de DMA. Para a placa do processador, este sinal é usado para desabilitar o barramento que estava de posse da CPU, e colocá-lo à disposição do controlador de DMA. Para os dispositivos de I/O conectados ao barramento de expansão esse sinal deve ser usado para desabilitar os decodificadores. Isto impede que um I/O responda, erroneamente, ao endereço de DMA (pois diz respeito à memória), já que vai estar presente um IOR ou IOW.

ALE (*Address Latch Enable*), saída: este é um sinal, originalmente gerado pelo controlador de barramento (8288), usado para indicar que existe um endereço válido no barramento. É ativado antes da validação dos endereços, porém só é desativado após os endereços estarem válidos. O flanco descendente (↓) deste sinal era usado como *strobe* para os latches que faziam a demultiplexação do barramento local do 8088, função esta não mais exercida a partir do 286. Como, no barramento de expansão, as linhas de endereços e dados já estão demultiplexadas, o sinal de ALE não tem serventia para demultiplexação. Porém, ele é um indicador de início de ciclo de barramento. Durante as operações de DMA, ele não é ativado.

TC (*Terminal Count*), saída: este é um sinal gerado pelo controlador de DMA (8237). Ele indica que um dos canais de DMA realizou o número de ciclos programado, e que terminou uma transferência de DMA. Como é um único sinal para os quatro canais, é necessário fazer uma lógica com os DACKi, por exemplo, gerando o sinal de contagem final para o canal i: TCi = [NOT (DACKi)] AND TC.

IOR (*I/O Read*), saída: este é um sinal gerado pelo controlador de barramento (8288), e indica aos dispositivos de I/O que o atual ciclo é um ciclo de leitura de I/O. O dispositivo endereçado deve responder colocando o dado no barramento pouco antes da subida do sinal IOR. Durante os ciclos de DMA, o sinal IOR é gerado pelo controlador de DMA (8287). Neste caso, o endereço presente no barramento diz respeito a uma posição de memória, e não a um dispositivo de I/O. O dispositivo de I/O é selecionado não por endereço, mas sim pelo sinal DACK.

IOW (*I/O Write*), saída: este é um sinal semelhante ao anterior, exceto por servir para escritas em um dispositivo de I/O. O dispositivo endereçado deve capturar (fazer um latch) o barramento de dados com o flanco ascendente (↑) desse sinal. A mesma afirmação é válida para os ciclos de DMA.

RST DRV (*Reset Driver*), saída: este sinal é mantido ativo durante a seqüência de energização do sistema. Ele permanece ativo até que as fontes de alimentação atinjam os níveis necessários. Além disso, é automaticamente ativado se alguma das fontes cair fora das especificações, e usado para inicializar todos os dispositivos conectados ao barramento de expansão, sendo sincronizado pelo 8284 (ou compatível).

DACK 0 – 3 (*DMA Acknowledge* de 0 até 3), saída:	esses sinais são gerados pelo controlador de DMA (8237) e indicam ao dispositivo de I/O que seu pedido de DMA foi aceito, e também indicam que o 8237 vai começar os ciclos correspondentes. O DACK 0 está presente no barramento original, apesar da ausência do DRQ0. Ele era usado para indicar um ciclo de DMA para *refresh* (e podia ser usado pelas palcas de memória para fazer seu *refresh*). No buffer implementado está disponível apenas o DACK3.
DACK 5-7 (*DMA Acknowledge* de 5 a 7), saída:	confirmação dos pedidos de DMA para os canais 5, 6 e 7. Eles trabalham de forma semelhante aos canais 1 e 3, exceto que a comunicação é feita a 16 bits. São ativados em nível baixo. No buffer implementado está disponível apenas o DACK5.
IRQ 2- 7 (*Interrupt Request* de 2 a 7), entrada:	estas são seis entradas que podem provocar interrupção mascarável na CPU. Esses sinais são levados diretamente ao controlador de interrupções (8259). A BIOS inicializa o 8259 de tal forma que a IRQ 0 é a de mais alta prioridade e a IRQ 7 a de mais baixa. Normalmente, são preparadas trabalhar com flanco ascendente (↑). Após o flanco ascendente, o sinal deve permanecer ativo até a chegada do sinal INTA. Do 286 em diante, a IRQ 2 foi substituída pela IRQ 9. No buffer implementado estão disponíveis as IRQs 3, 5, 7 e 9.
IRQ 10, 11, 12, 14 e 15 (*Interrupt Request*), entradas:	são os pedidos de interrupção IRQ 10, 11, 12, 14 e 15. De certa forma, são idênticos aos pedidos IRQ 3 a 7, só que alimentam um segundo controlador de interrupções (8259), cascateado com o primeiro (via IRQ 2). São ativados pelo flanco de subida. No buffer implementado está disponível apenas a IRQ 10.
DRQ 1 – 3 (*DMA Request* de 1 até 3), entrada:	através dessas linhas, os dispositivos de I/O podem pedir DMA. Se um dispositivo necessita fazer uma operação de DMA, ele levanta uma dessas linhas. O pedido (DRQi) vai direto para o controlador de DMA (8237), onde ele é priorizado antes de ser aceito. A BIOS inicializa o 8237 de forma que o DRQ 0 é o de mais alta prioridade. A linha DRQ 0 não está disponível no barramento original do sistema, pois ela era usada para fazer o *refresh* das memórias dinâmicas. O pedido DRQ i deve ser apagado quando chega a confirmação DACK i. No buffer implementado está disponível apenas o DRQ 3.
DRQ 5 – 7 (*DMA Reqwuest* de 5 a 7), entrada:	são os sinais de pedido de DMA para os canais 5, 6 e 7. Eles trabalham de forma semelhante aos canais 1 e 3, exceto que a comunicação é feita a dezesseis bits. São ativados em nível alto. No buffer implementado está disponível apenas o DRQ 5.
+5 V DC:	regulagem de ±5%, está presente em dois pinos do conector de expansão.

-5 V DC: regulado para ±10%.

+12 V DC: regulado para ±5%.

-12 V DC: regulado para ±10%.

GND: referência de terra, presente em três pinos do conector de expansão.

INTERFACE: este sinal tem como finalidade habilitar/desabilitar o 74LS245 para liberar/bloquear o barramento do PC. Através dos sinais IOR e IOW o 74LS245 controla o sentido de dados pelo barramento.

CLK (*CLOCK*), saída: este sinal é obtido a partir do sinal OSC, mediante um divisão por três, gerando então o sinal de 4,77 MHz (8886/88) ou 8 MHz (286, 386, 486, etc.). O sinal CLK não é simétrico, apresentando um ciclo de carga de 33%, ou seja, 1/3 em alto e 2/3 em baixo. Ele é muito bem sincronizado com os ciclos de leitura e escrita, e por isso, pode ser usado para a temporização de todos os outros sinais que o façam necessário.

A figura 23 ilustra o circuito montado para a realização dos experimentos.

Figura 23 - Diagrama esquemático do circuito utilizando o microcontrolador PIC16C73A.

A figura 24 mostra uma foto do hardware.

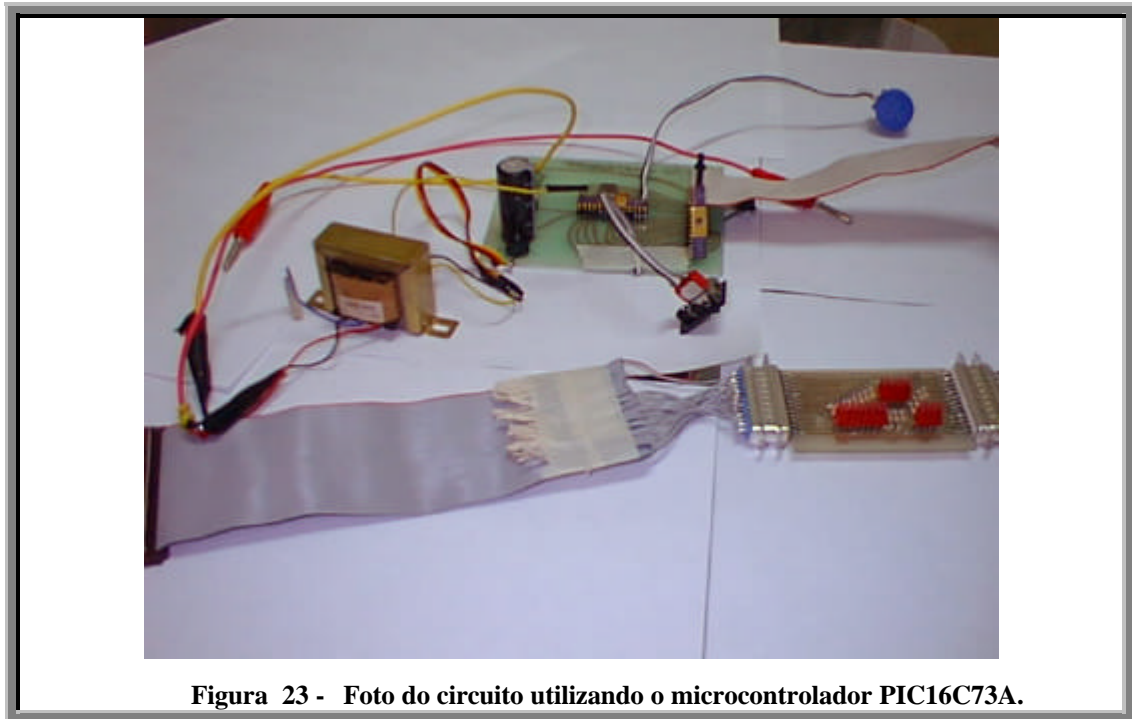



Figura 23 - Foto do circuito utilizando o microcontrolador PIC16C73A.

Este circuito é conectado através de "*flat cable*", adaptando a saída D25 da placa "PIC16C73A" ao conector de 50 pinos da placa "*buffer*". Os pedidos de DMA e IRQ são realizados através do software interno ao PIC16C73A mostra a seguir:

```
#include <DMA_irq.H>
#define A9    7
#define A8    6
#define AEN   5
#define IOR   4
#define INTER 3
#define DACK  2
#define DRQ   1
#define IRQ   0
#define TIPO  2

envia_dado_dma(int dado)
{
  bit_set(port_C,DRQ);                     // Faz o pedido de DMA
  set_tris_b(0x00);                        // porta B configurada para saida.
  port_b = dado;                           // dado colocado no barramento de DADOS
  bit_clear(port_C,DRQ);                   // Retira o sinal de DRQ (ciclo de DMA em
                                           // andamento)

  do
  {
    // Este loop espera que o sinal:
    // - DACK va para nivel alto;
  } while ( (!bit_test(port_C,DACK)) );

  set_tris_b(0xFF);                        // porta B configurada para leitura.
}

envia_dado_irq(int dado)
```

```
{
  bit_set(port_C,IRQ);                          // Faz o pedido de IRQ

  do
  {
    // Este loop espera que os sinais:
    // - A9  va para nivel alto;              Isto significa uma leitura no periferico
    // - A8  va para nivel alto;              conectado ao endereco 0x300
    // - IOR va para nivel baixo
  } while ( (((!bit_test(port_C,A9)) || (!bit_test(port_C,A8))) ||
             bit_test(port_C,IOR)) );

  bit_clear(port_C,IRQ);                       // Retira o sinal de IRQ pois ja esta
                                               // sendo atendida
  bit_clear(port_C,INTER);                     // habilita a interface para leitura
  set_tris_b(0x00);                            // porta B configurada para saída.
  port_b = dado;                               // dado colocado no barramento de DADOS

  do
  {
    // Este loop espera que os sinais:
    // - IOR va para nivel alto;
  } while ( (!bit_test(port_C,IOR)) );

  bit_set(port_C,INTER);
  set_tris_b(0xFF);
}

main()
{
  unsigned int Dado_AD;

  setup_counters(RTCC_INTERNAL,RTCC_DIV_2);
  setup_port_a(RA0_RA1_RA3_ANALOG);
  setup_adc(ADC_CLOCK_DIV_8);
  set_adc_channel(0);
  set_tris_b(0xFF);
  set_tris_c(0xFD);
  delay_us(10);


  bit_clear(port_C,DRQ);

  Dado_AD = 0;

  do
  {
    // Faz uma leitura do conversor AD e envia para transferencia para o micro
    // Dado_AD = read_adc();

    // Testar se o bit A2 e 1 - se True transferencia por DMA - se False transferencia
    // por IRQ

    if (bit_test(port_A,TIPO))
    {
       envia_dado_dma(Dado_AD);
       Dado_AD = Dado_AD + 1;
       delay_us(100);
    }
    else
    {
       envia_dado_dma(Dado_AD);
       Dado_AD = Dado_AD + 1;
       delay_us(100);
    }
  } while(TRUE);
}
```

5 – SOFTWARE PARA DMA E INTERRUPÇÕES

Com base na teoria apresentada nos itens 2 e 3, foi montado o programa para a aquisição dos dados através de DMA e Interrupções. O programa encontra-se devidamente comentado e sua listagem encontra-se a seguir:

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <stdlib.h>
#include <bios.h>

#define RegMode 0xB     // Define os modos de operacao para cada um dos
         // 4 canais de DMA de 8 Bits (DMA's  0, 1, 2 e 3)
         // Palavra de comando para o RegMode
         // D7 D6 D5 D4 D3 D2 D1 D0
         // Para se conseguir a palavra de controle
         // para o RegMode basta aplicar a logica
         // OR para os bits de dados abaixo
         // Ex.: Byte | IEnd | HAI | Wri | DMA3 = 0x57 ou 87 (dec)

// Define o Registrador para comandos - (tipo de DMA a ser realizado)
#define RegComm 0x8

// Os Bits D1 e D0 definem o canal de DMA que sera selecionado
#define DMA0 0x00
#define DMA1 0x01
#define DMA2 0x02
#define DMA3 0x03

// Os Bits D3 e D2 definem o tipo de operacao que sera realizada
#define Ver  0x00       // Verificao
#define Wri  0x04       // Escrita na memoria
#define Rea  0x08       // Leitura na memoria
#define Ilg  0x0C       // Operacao ilegal

// O Bit D4 Habilita/Desabilita a auto inicializacao
#define DAI 0x00        // Desabilita a Auto Inicializacao (DAI)
#define HAI 0x10        // Habilita a Auto Incializacao (HAI)

// O Bit D5 seleciona o incremento/decremento de endereco
#define IEnd 0x00       // Seleciona incremento de endereco
#define DEnd 0x20       // Seleciona decremento de endereco

// Os Bits D7 e D6 definem o modo de operacao do DMA (byte, burst, etc...)
#define Dema  0x00      // Seleciona o modo de Demanda
#define Byte  0x40      // Seleciona o modo Byte (um byte de cada vez)
#define Bloco 0x80      // Seleciona o modo Burst (rajada)
#define Casca 0xC0      // Seleciona o modo cascata
```

```
// Definicao dos enderecos de programacao
#define EndBaseDMA3 0x6   // Define o Endereco base e corrente para o DMA 3
#define ConBaseDMA3 0x7   // Define o Contador base e corrente para o DMA 3
#define RegPagiDMA3 0x82  // Define o Endereco do registrado de pagina DMA 3
#define Reseta_FF   0xC   // Define o Endereco de reset do Flip Flop

#define Single_Mask 0xA // Define o Endereco do registrador de mascara simples
         // para 4 canais de DMA de 8 Bits (DMA's  0, 1, 2 e 3)
         // Palavra de comando para Single_Mask
         // DX DX DX DX DX D2 D1 D0
         // Para se conseguir a palavra de controle
         // para o SingleMask basta aplicar a logica
         // OR para os bits de dados abaixo
         // Ex.: HSM | SM_DMA3 = 0x03 ou 3 (dec)

// Os Bits D1 e D0 definem o canal de DMA que sera selecionado
#define SM_DMA0 0x00
#define SM_DMA1 0x01
#define SM_DMA2 0x02
#define SM_DMA3 0x03

// O Bit D2 define se a mascara esta habilitada ou nao
#define HSM 0x00   // Neste caso o canal funciona - apaga a mascara
#define DSM 0x04   // Neste caso o canal nao funciona - ativa a mascara

// Enderecos para I/O com a placa de DMA
#define PlacaDMA 0x306   // Este I/O requisita o DMA da placa
#define PlacaIRQ 0x305   // Este I/O requisita o IRQ da placa
#define PlacaIO  0x300   // Endereco da porta de leitura de dados

// Definicao para interrupcoes
#define IRQ05 0xD
#define IRQ10 0x72
#define IRQ11 0x73
#define CRTL_IRQ 0x21

// Declaracao dos prototipos das funcoes utilizadas
void Inicializa_DMA(unsigned int Bytes, unsigned char Modo);
void Recebe_Dados_DMA(unsigned int Bytes);
void Recebe_Dados_IRQ(unsigned int Bytes);
void Imprime_Dados(unsigned int Bytes);

// Declaracao das variaveis globais do programa
unsigned char *Dado,*Dados;   // Esta variavel contem o endereco da memoria
               // onde sera armazenado os dados recebidos atraves
               // de DMA.
unsigned char *PDados;
unsigned int *Cont_irq;
unsigned char NomeArq[30];

void interrupt ( *antiga)(...);
```

```
void interrupt Le_Dado_300(...)
{
   *PDados = (unsigned char) inportb(PlacaIO);
   PDados++;
   *(unsigned int *) Cont_irq = *(unsigned int *) Cont_irq + 1;
   outportb(0x20,0x20);
}

void Inicializa_DMA(unsigned int Bytes, unsigned char Modo)
{
  unsigned long int Seg_Dado,   // O endereco da variavel Dado sera dividido
          Off_Dado,   // em Segmento (64 kB) e Offset (definicao
          Endere20,   // de paginas).
          Endere16,
          PaginaEn;

  Seg_Dado = FP_SEG(Dado);
  Off_Dado = FP_OFF(Dado);
  Endere20 = (Seg_Dado<<4) + Off_Dado;
  Endere16 = Endere20 & 0xFFFF;
  PaginaEn = Endere20 >> 16;

  // Mascara o canal de DMA 3, para que ele possa ser programado
  // com as novas configuracoes
  outportb(Single_Mask, (DSM | SM_DMA3));

  // Envia comando programando o DMA - (Hab. DMA, Nao Memoria, etc...)
  outportb(RegComm, 0x00);

  // Atraves deste comando e resetado o flip flop que controla
  // a programacao dos bytes LSB e MSB tanto para o endereco Base,
  // quanto para o Contador.
  outportb(Reseta_FF,0x00);

  // Atraves deste comando e programado o Registrador de Modo do 8237
  // informando qual o canal de DMA, se e escrita ou leitura, se
  // os enderecos serao incrementados ou decrementados e o tipo de
  // transferencia (byte, burst, etc...)
  outportb(RegMode,(Modo | IEnd | HAI | Wri | DMA3));

  // Este comando informa ao controlador DMA o endereco (composto por
  // Seg_Dado e Pagina ) no qual devera ser lido ou escrito o dado
  // na memoria
  outportb(EndBaseDMA3, Endere16 & 0xFF); // joga fora MSB e guarda LSB
  outportb(EndBaseDMA3, Endere16 >> 8);   // joga fora LSB e guarda MSB
  outportb(RegPagiDMA3, PaginaEn);      // define a pagina

  // Define o numero de bytes que serao lidos ou escritos de cada vez
  // Como a transicao de TC ocorre de 0x0000 para 0xFFFF e necessario
  // decrementar a variavel Bytes (contador) de 1.
  Bytes--;
  outportb(ConBaseDMA3, Bytes & 0xFF);
  outportb(ConBaseDMA3, Bytes >> 8);
```

```
}

void Recebe_Dados_DMA(unsigned int Bytes)
{
  unsigned char TC, DP;
  unsigned long Tempo;

  clrscr();
  TC =  ((unsigned char) inportb(0x08)) & (0x08);
  DP =  ((unsigned char) inportb(0x08)) & (0x80);

  PDados = Dados;
  *Dados = 33;

  gotoxy(3,1);
  printf("Condicoes iniciais de TC e DP");
  gotoxy(3,2);
  printf("TC = %x",TC);
  gotoxy(3,3);
  printf("DP = %x",DP);
  gotoxy(3,4);
  printf("Pressione qualquer tecla..." );

  getch();
  clrscr();
  gotoxy(3,1);
  printf("DMA em progresso");

  Tempo = biostime(0,0L);

  outportb(Single_Mask, (HSM | SM_DMA3));

  do
  {
    TC = ((unsigned char) inportb(0x08)) & (0x08);
    DP = ((unsigned char) inportb(0x08)) & (0x80);
  }
  while ( (!(TC&0x08)) && (!kbhit()) );

  Tempo = biostime(0,0L) - Tempo;

  gotoxy(3,2);
  printf("Tempo gasto para realizar o DMA: %ld",Tempo);
  gotoxy(3,3);
  printf("TC = %x",TC);
  gotoxy(3,4);
  printf("DP = %x",DP);

  outportb(PlacaDMA,0x00);
  outportb(Single_Mask, (DSM | SM_DMA3));
  gotoxy(3,5);
  printf("Fim de DMA");
}
```

```
void Imprime_Dados(unsigned int Bytes)
{
  int i;
  FILE *fp;

  clrscr();

  if ((fp = fopen(NomeArq,"wt"))==NULL)
  {
     printf("Nao foi possivel abrir o arquivo para escrever os dados" ) ;
     return;
  }

  PDados = Dado;

  printf("Dados adquiridos do Hardware\n");
  fprintf(fp,"Dados adquiridos do Hardware\n" );
  for(i=0;i<Bytes;i++,PDados++)
  {
     printf("%d ", (int) *PDados);
     fprintf(fp,"%d ", (int) *PDados);
  }
  fclose(fp);
}

void Recebe_Dados_IRQ(unsigned int Bytes)
{
   unsigned int Mascara_IRQ,i,j;

   Cont_irq = (unsigned int *) malloc(sizeof(int));
   *(unsigned int *) Cont_irq = 0;
   antiga = getvect(IRQ05);              // Guarda o antigo vetor de interrupcao
   setvect(IRQ05,Le_Dado_300);           // Aponta para a nova rotina de interrupcao

   Mascara_IRQ = (unsigned char) inportb(CRTL_IRQ);  // Le o conteudo de OCW1
   outportb(CRTL_IRQ,Mascara_IRQ&(0xDF));         // Habilita a IRQ 5
   PDados = Dado;
   outportb(0x20,0x20);

   outportb(0x306,0x00);
   outportb(0x306,0x00);

   do { } while ( ( (*(unsigned int *) Cont_irq) < Bytes) && (!kbhit()) );

   outportb(0x306,0x00);
   outportb(CRTL_IRQ,Mascara_IRQ);
   setvect(IRQ05,antiga);
}

void main(void)
{
  unsigned int Tipo, N_Bytes, i;
```

```
  clrscr();

  printf("Tipo de transferencia ( 0-DMA / 1-IRQ )\n");
  scanf("%d",&Tipo);
  printf("Quantos bytes serao recebidos na transferencia?\n");
  scanf("%d",&N_Bytes);
  printf("Nome do arquivo de saida?\n");
  scanf("%s",NomeArq);

  Dado = (char *) malloc(N_Bytes * sizeof(char));
  Dados = (char *) malloc(N_Bytes * sizeof(char));

  for(i=0,PDados=Dado;i<N_Bytes;i++,PDados++) *PDados = 170;

  if (Tipo==0)
  {
     Inicializa_DMA(N_Bytes,Byte);
     Recebe_Dados_DMA(N_Bytes);
     printf(" Pressione qualquer tecla para imprimir os dados adquiridos" );
     getch();
     Imprime_Dados(N_Bytes);
  }
  else
  {
     Recebe_Dados_IRQ(N_Bytes);
     printf(" Pressione qualquer tecla para imprimir os dados adquiridos" );
     getch();
     Imprime_Dados(N_Bytes);
  }
  getch();
}
```

6 – CONCLUSÃO (para cumprir tabela)

Através deste pequeno projeto (mas como foi difícil de implementar, 2 semanas de trabalho árduo – onde estava os 30 segundos???) pode-se desenvolver hardware e software dedicados ao controle de DMA e Interrupções. Adicionalmente, pelo fato de se trabalhar com o PIC16C73A, foi possível aos membros do grupo adquirir um pouco conhecimento sobre este microcontrolador. Assim verificou-se detalhadamente as etapas de implementação do hardware e o que deveria ser programado para o seu funcionamento, tanto no microcontrolador quanto no PC.

7 – BIBLIOGRAFIA

[1] PC - Um Guia Prático de Hardware e Interfaceamento – Ricardo Zelenovsky, Alexandre Mendonça – Editora Interciência.

[2] PC e Periféricos – Um Guia Completo de Programação - – Ricardo Zelenovsky, Alexandre Mendonça – Editora Ciência Moderna.

[3] PIC16C7X Data Sheet – 8 Bit CMOS Microcontrollers with A/D Converter – MICROCHIP.

[4] PCB, PCM And PCW Reference Manual – CCS – Custon Computer Services, Inc. – March 1998.

[5] Materiais diversos coletados da INTERNET.